embedded-projects.net
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Eine Open-Source Zeitschrift
zum Mitmachen!

Leseratten Ahoi

[PROJECTS]

- GPS Logger zum Selberbauen
- Web-basiert Messen, Steuern und Regeln
- Ein EKG-Simulator
- Touchpad - die andere Bedieneinheit
- GNUBLIN Teil 2 - Wir bauen einen Rechner



Augsburg Ahoi!

Der Frühling geht der Sommer kommt

Embedded Projects Journal - Ausgabe No. 9

Meister der Herzen

Ein gähnender leichter Sonntag Nachmittag. Die Stimmung passt - was will man mehr. Anpfiff - Daumen drücken! Nie wieder Liga 2? Bange Minuten, Bange Zeit - Halbzeit. Kein Tor, kein Ruhm keine Liga 1. Doch dann kurz vor dem Schluss ein Schuss! Die Stadt tobt. Es hupt, und jault! Wir hams geschafft!

Augsburg in der Bundesliga! Klingt komisch? Ist aber so :-). In diesem Sinne werden wir die nächsten Monate im Zeichen des Fußballs genießen und freuen uns ab August auf all die großen Städte die uns dank der Liga Nr 1 besuchen kommen.

Wir freuen uns ein kleiner Teil dieser Stadt zu sein. Mehr von Augsburg gibts im nächsten Heft.

Dankeschön an unsere Autoren:
Aktuell erhalten wir immer mehr zugesendete Artikel! Wir freuen uns über all die spannenden Texte. Bitte weiter so!

Im nächsten Heft sind u.a. geplant:

- Leiterplatte Teil 2
- Touchbox - ein anderes Musikinstrument
- Ein FPGA Softcore

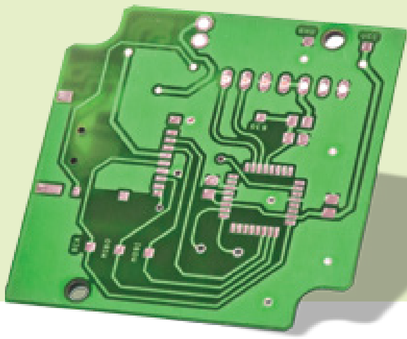
Benedikt Sauter
sauter@embedded-projects.net

und das embedded projects Team

GPS Logger Mini

Eine Bauanleitung

Martin Matysiak <mail@k621.de>



Einleitung

Nachdem ich mit meinem GPS Logger nun schon seit zwei Jahren erfolgreich diverse Strecken und Touren aufzeichne, wurde es Zeit für eine Neuauflage des Gerätes. Vor allem die Größe war auf Radfahrten eher hinderlich, hinzu kam die störanfällige Kabelverbindung des GPS-Moduls, die den alltäglichen Belastungen nicht immer stand hielt. Ein weiterer Aspekt beim Entwurf der zweiten Version, der mir wichtig war, ist eine leichte Nachbaumöglichkeit des Loggers. Aus diesem Grund verwende ich nur Bauteile, die bei den bekannten Distributoren erhältlich sind. Die Platine ist exakt auf das Gehäuse abgestimmt, sodass eine stabile Konstruktion gewährleistet ist.

Nachfolgend eine Auflistung der wichtigsten Änderungen im Vergleich zum Vorgänger:

- Deutlich kleiner (Außenmaße: 5cm x 5cm x 2cm)
- Integrierte Ladeschaltung
- Keine Schalter/Tasten - einlegen der SD-Karte schaltet das Gerät ein, herausnehmen schaltet es aus
- Aus Platzgründen verzicht auf ein FAT-Dateisystem auf der SD-Karte - Stattdessen Einsatz des selbstentwickelten „NoFS“ (No FileSystem)
- Kein Schutz vor Tiefentladung. In der Regel sind LiIon Akkus durch eine interne Schaltung vor Tiefentladung geschützt. Sollte dies nicht der Fall sein bitte den Akku regelmäßig laden
- Geschätzte Akkulaufzeit: 10 bis 15 Stunden
- Doppelseitiges Layout, GPS-Modul befindet sich direkt auf der Platine (höhere Stabilität)

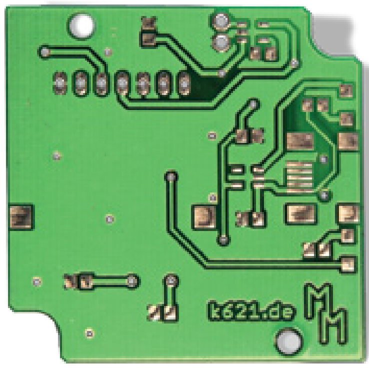


Abb. 1: Doppelseitige Platine

Eine Beispielroute, die mit dem gLogger Mini aufgezeichnet wurde, kann auf Google Maps betrachtet werden. Einige Kurven sind dort sehr geschnitten, das liegt an der maximal möglichen Anzahl an Koordinaten bei Google Maps [1], weswegen der Pfad dementsprechend vereinfacht werden musste.

Die Platine

Die Platine ist von den Maßen her nach den Vorgaben des Gehäuses gestaltet. Mit Hilfe der zwei Bohrungen lässt es sich somit passgenau montieren. Ich habe ein doppelseitiges Layout erstellt, bei dem fast ausschließlich SMD-Bauteile zum Einsatz kommen, um möglichst viel Platz zu sparen. Das GPS-Modul ist fest auf der Platine angebracht, lediglich der Akku wird über ein kurzes Kabel zur Platine herangeführt. Alle Bauteile passen ohne große Probleme in das Gehäuse, näheres dazu in den folgenden Abschnitten.

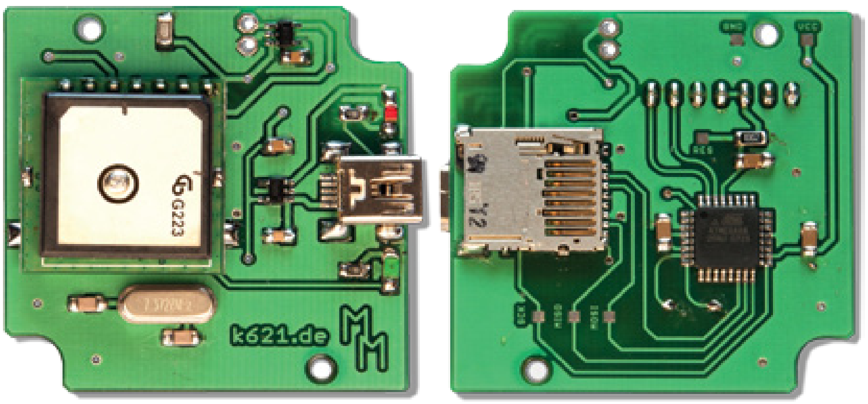


Abb. 2: Bestückung der Platine

Der Schaltplan sowie das Layout wurde mit Eagle erstellt, der Projektordner [2] kann im „Downloads“-Bereich abgerufen werden. Ober- und Unterseite der Platine sind in Abbildung 1,7 und 8 dargestellt, der Schaltplan des gLogger Mini in Abbildung 10.

Aus Platzgründen wurde auf viele Extras des vorherigen GPS Loggers verzichtet, so wird die Akkumulatorspannung nicht mehr kontrolliert und die unverbundenen Pins sind auch nicht mehr zu Steckern herausgeführt. Auch auf einen klassischen ISP-Programmierstecker habe ich verzichtet. Auf der Unterseite der Platine sind allerdings alle 6 notwendigen Kontakte als große Lötpads freigelegt, an denen man Kabel zu einem passenden Stecker anlöten kann (siehe Bauanleitung). Da der Mikrocontroller in der Regel nur einmalig oder nur sehr selten umprogrammiert werden muss, halte ich diesen Mehraufwand für vertretbar.

Abb. 3: Einbau in das Gehäuse



Zusammenbau

In diesem Abschnitt möchte ich eine Schritt für Schritt Anleitung zum Zusammenbau des „gLogger Mini" anbieten, sodass jeder einen Nachbau schafft.

In der Tabelle „Benötigte Bauteile" (Tab. 1) befinden sich nicht nur die Bauteile für die Platine, sondern auch alle anderen Bauteile, die für den Zusammenbau notwendig sind. Der dort aufgeführte Gesamtpreis entspricht somit ziemlich genau dem, was für einen Nachbau ausgegeben werden muss (zzgl. Versandkosten und MwSt. bei Farnell). Das Einzige, was dort nicht aufgelistet ist, sind zwei Schrauben zum Fixieren der Platine im Gehäuse. Diese sollten sich allerdings im Haushalt oder im nächsten Baumarkt kostengünstig auftreiben lassen. Näheres dazu in der Anleitung.

Als Farnell-Alternative sei hbe-shop.de genannt, bei dem auch von Privat bestellt werden kann. Der Shop nutzt die Farnell-Bestellnummern.

Da fast ausschließlich SMD-Bauteile verwendet werden, sollten generelle Lötkenntnisse vorhanden sein. Diese Anleitung ist lediglich meine Vorgehensweise beim zusammenbau des Geräts. Es besteht kein Zwang, die selbe Reihenfolge einzuhalten. Mit dieser Anleitung möchte ich lediglich auf ein paar Feinheiten bei der Montage hinweisen.

Schritt 1: Fangen wir mit der Oberseite der Platine an. Diese ist relativ unkritisch, alle SMD Bauteile können schon aufgelötet werden. Lediglich mit den beiden größten Bauteilen sollte man noch warten. Dies wäre zum einen der Quarz, der nach dem Verlöten mit zwei „Löthügeln" auf der Unterseite das Befestigen des Mikrocontrollers stören könnte. Zum anderen handelt es sich um das GPS-Modul. Man kann es eigentlich auch jetzt schon auflöten, wenn man allerdings nachher noch selber am Programm für den Mikrocontroller arbeiten möchte, sind freie UART-Leitungen zwecks Debugging sehr hilfreich. Aus diesem Grund habe ich mir das

Bauteil	Beschreibung	Wert	Bauform	Distributor	Preis
C1	Keramik-Kondensator	1µF	0805	Alle	0,07 €
C2	Keramik-Kondensator	1µF	0805	Alle	0,07 €
C3	Keramik-Kondensator	10nF	0805	Alle	0,05 €
C4	Keramik-Kondensator	100nF	0805	Alle	0,05 €
C5	Keramik-Kondensator	100nF	0805	Alle	0,05 €
C6	Keramik-Kondensator	22pF	0805	Alle	0,05 €
C7	Keramik-Kondensator	22pF	0805	Alle	0,05 €
C8	Keramik-Kondensator	100nF	0805	Alle	0,05 €
IC1	Low-Drop Spannungsleiter 3.3V	LP2985	SOT23-5L	F	0,56 €
IC2	Laderegler	MAX1555	SOT23-5L	F	2,41 €
IC3	Mikrocontroller	ATMega88	TQFP32	F	3,56 €
IC4	MicroSD-Kartenslot	Hirose DM3B	SMD	F	2,23 €
IC5	GPS-Modul	ST22	-	P	20 €
LED1	SMD Leuchtdiode	ROT	1206	Alle	0,08 €
LED2	SMD Leuchtdiode	GRÜN	1206	Alle	0,08 €
Q1	Quarz	7,3728MHz	HC49/S	Alle	0,18 €
R1	Widerstand	470Ω	0805	Alle	0,10 €
R2	Widerstand	10kΩ	0805	Alle	0,10 €
R3	Widerstand	10kΩ	0805	Alle	0,10 €
R4	Widerstand	470Ω	0805	Alle	0,10 €
X1	USB Buchse	Mini-B 5 pol.	SMD	R	0,24 €
-	Akku	600mAh	iPod Mini	R	5,30 €
-	Gehäuse	Hammond 1551R	5x5x-2cm³	C	1,67 €
-	Platine			Ich!	10 €
	Gesamtpreis				**47,15 €**

Tab. 1: Benötigte Bauteile

Modul für den Schluss aufgespart.

Tipp: Bei Bauteilen mit vielen Pins (z. B. die USB-Buchse) kann man ruhig viel Lötzinn verwenden, auch wenn Lötbrücken entstehen. Das überschüssige Zinn kann man dann mit einer Entlötlitze entfernen, anschließend hat man sehr gut aussehende Lötstellen. Das verfahren erläutere ich allerdings gleich noch einmal mit Bildern beim Mikrocontroller (siehe Abbildung 5).

Tipp: Wenn man nur SMD-1206-Bauteile auf Lager haben sollte, bekommt man diese auch ganz gut auf die 0805er Pads. Ein Beispiel dafür findet sich in der Abbildung 4 (die 22pF Kondensatoren in der Nähe des Quarzes).

Schritt 2: Nun sind SD-Slot und Mikrocontroller dran. Da beide sehr geringe Pinabstände besitzen, ist das Verfahren zum Verlöten bei beiden Teilen sehr ähnlich. Ich verwende die Methode mit einer Entlötlitze, welche im nachfolgenden Bild (Abbildung 5) dargestellt ist. Das Verfahren ist relativ simpel:

a) Bevor das Bauteil platziert wird, zunächst die Pads mit etwas Lötpaste bedecken, sodass der Lötvorgang später vereinfacht wird (optional). Anschließend wird das Bauteil ausgerichtet und an zwei Pads (am besten gegenüberliegend) fixiert

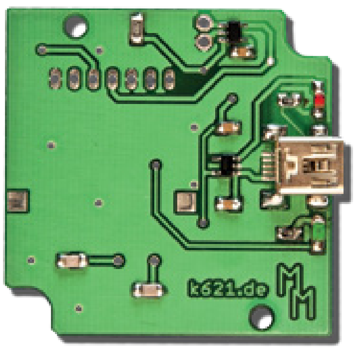


Abb. 4: Platine

b) Eine Kante wird großzügig mit Zinn bedeckt und verlötet

c) Mit einer Entlötlitze wird so lange überschüssiges Zinn wieder entfernt, bis keine Zinnbrücken mehr vorhanden sind

d) Wiederholung von Schritt b und c bei den verbleibenden Kanten

Das Ergebnis ist eine sehr gut aussehende Lötstelle mit optimalem Kontakt.

Schritt 3: Die kritischsten Bauteile befinden sich nun auf der Platine. Was noch fehlt, sind einige Kondesatoren, ein Widerstand sowie (falls vorhin weggelassen) der Quarz und das GPS-Modul. Nach dem erfolgreichen Löten sollte die Platine wie beim nebenstehenden Bild aussehen.

Schritt 4: Nun muss noch der Akkumulator mit der Platine verbunden werden. Ausgeliefert wird er mit einem Stecker - eigentlich war eine passende Buchse geplant, leider haben wenige Millimeter für diesen Plan gefehlt. Aus diesem Grund muss der Stecker behutsam entfernt werden (die Kabel nicht davor abschneiden, sonst sind sie zu kurz!). Schaut man auf die Unterseite der Platine (die Seite mit µC und SD-Slot), so sollten die Kabel von links an die Vias herangeführt werden, damit alles gut in das Gehäuse passt. Hierbei muss man ein wenig probieren, bis es passt.

Schritt 5: Bevor der Mikrocontroller programmiert wird, sollte man zunächst den Akkumulator vollständig aufladen (wer weiß, in welchem Zustand er ankommt..). Hierfür einfach das Gerät per USB entweder an den Computer oder an ein Netzteil mit USB-Buchsen-Ausgang stecken. Wenn alles geklappt hat, leuchtet jetzt die rote LED, die anzeigt, dass der Akku geladen wird. Zeit für eine kleine Pause :-)

Schritt 6: In der Zwischenzeit kann man sich natürlich auch den Programmieradapter basteln. Ich habe hierfür einen 2x3-Pin Stecker auf eine Lochraster Platine gebracht und anschließend an jeden Pin ein Stück Draht angelötet, welcher nachher mit den entsprechenden Pads auf der Platine verbunden werden muss. Das Foto Abbildung 9 zeigt meine Konstruktion. Es gilt Pinbelegung aus Abbildung 6.

Schritt 7: Sobald der Akku geladen und der Programmieradapter verbunden ist, wird es Zeit, das Gerät zu programmieren. Damit das Gerät Strom bekommt, muss während dieses Vorgangs eine MicroSD-Karte eingelegt sein. Neben dem Programm müssen auch die Fuses richtig eingestellt werden. Ich verwende hierfür das Programm „avrdude" mit folgenden Parametern [3]:

```
-U lfuse:w:0xdc:m -U
hfuse:w:0xdf:m
```

Ist das Gerät richtig programmiert, dürfe die grüne LED nach einem Initialisierungsvorgang mit einer Frequenz von 1Hz aufblitzen (wenn man nichts an den eingehenden NMEA-Kommandos verändert hat). Sollte dies der Fall sein, kann man nun den Programmieradapter wieder entfernen und die Platine im Gehäuse verschrauben.

Schritt 8: Leider liefert das Gehäuse nur Schrauben mit, um den Deckel zu befestigen, nicht aber für die inneren zwei Ge-

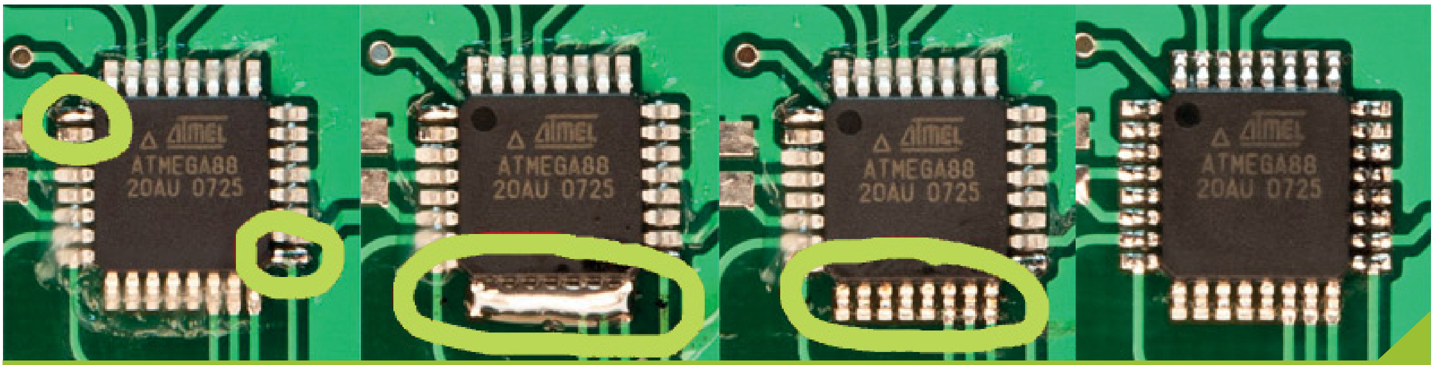


Abb. 5: Auflöten der Mikrocontroller

winde zur Befestigung der Platine. Diese muss man entweder zu Hause oder im nächsten Baumarkt suchen. Bei meinem Prototypen habe ich 2x12-Holzschrauben verwendet. Diese waren minimal zu lang! Ich würde also **2x10 oder 2x8** Schrauben empfehlen. Wichtig ist, dass sowohl die äußeren als auch die inneren Schrauben **nicht zu fest** angeschraubt werden dürfen. Ansonsten herrscht ein zu großer Druck auf den Kartenslot und die Karte verklemmt sich beim Einlegen oder Herausnehmen. Nach dem Anschrauben also immer prüfen, ob die Karte noch gut passt. Auf einer Seite des Gehäuses muss man etwas vom Plastik wegschneiden, damit Zugang zum USB-Port und zum Speicherkarten-Slot gegeben ist (siehe Abbildung 3).

Software AVR

Die Software für den Mikrocontroller besteht aus mehreren einzelnen Modulen, die in der Hauptdatei zusammengesetzt und abgefragt werden. Der generelle Ablauf des Programms sieht wie folgt aus:

- NMEA-String vom GPS-Modul holen
- Wenn GPS Fix, dann NMEA-String auf SD-Karte schreiben
- Einmal als optisches Feedback die LED aufblitzen lassen
- Schlafen legen, auf den nächsten Datenempfang per UART warten

Es ist zu beachten, dass die LED bei jedem eingehenden NMEA-Kommando blinkt, und nicht etwa bei jedem Schreibvorgang. Sollte die SD-Karte also trotz mehrfachen Blinkens leer sein, so hatte das Modul höchstwahrscheinlich keinen GPS-Fix. Während der Initialisierung der Module leuchtet die grüne LED dauerhaft. Wenn sie anschließend mit einer relativ hohen Frequenz blinkt (um genau zu sein 5Hz), so konnte entweder die SD-Karte oder das GPS-Modul nicht initialisiert werden. Die rote LED ist nicht mit dem Mikrocontroller verbunden und dient nur zum Anzeigen des Ladevorgangs (leuchtet sie bei eingestecktem USB-Kabel, so wird der Akku momentan geladen. Sobald die LED erlischt, ist der Ladevorgang beendet und der Akku aufgeladen).

Standardmäßig werden nur $GPGGA-Meldungen aufgezeichnet. In der Datei *modules/gps.c* lässt sich dies in der Funktion *gps_init()* allerdings leicht verändern. Man muss lediglich die Werte der Variable *commands* (Z. 31) ändern:

```
unsigned char commands[8]
= {
    0x01, //GGA 1Hz
    0x00, //Keine GSA
    0x00, //Keine GSV
    0x00, //Keine GLL
    0x00, //Keine RMC
    0x00, //Keine VTG
    0x00, //Keine ZDA
    0x00}; //In SRAM&FLASH
```

Der Wert gibt an, in welchem Zeitintervall (in Sekunden) das Kommando vom GPS-Modul ausgegeben werden soll. Der Wert 0 deaktiviert die jeweilige Meldung.

Sollten andere Kommandos gewünscht sein, so ist es ratsam bzw. notwendig, die Prüfung nach einer validen Positionsmeldung abzuschalten. Hierfür muss man folgende Codezeilen in der Hauptschleife (gLogger.c, Z. 95-97):

```
if(gps _
getNmeaSentence(nmeaBuf,
128))
{ nofs _
writeString(nmeaBuf);
}
```

auf folgende abändern:

```
gps _
getNmeaSentence(nmeaBuf,
28);
nofs _ writeString(nmeaBuf);
```

und schon wird jegliches eingehende NMEA-Kommando auf die Speicherkarte geschrieben. Es gibt noch einige andere Konfigurationsoptionen für das ST22 GPS-Modul. Diese sind in der Application Note AN0003 des Skytraq Venus6

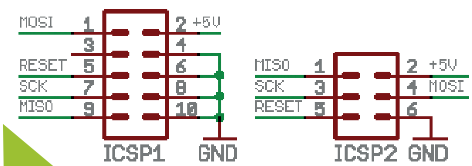


Abb. 6: Benötigte Bauteile

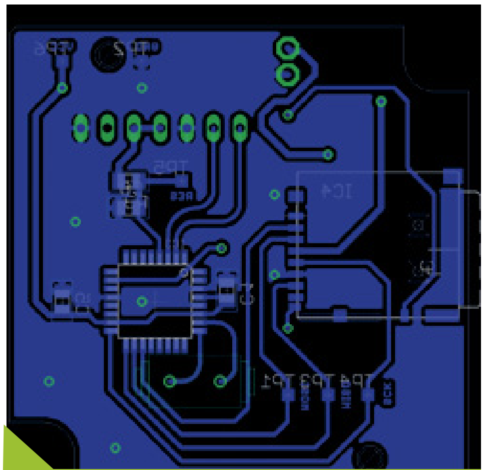


Abb. 7: Bottom Layer

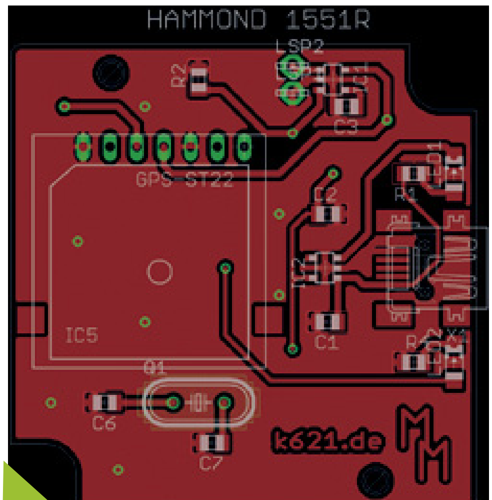


Abb. 8: Top Layer

Chipsatzes [4] aufgelistet.

Die Befehlsbibliothek für das *NoFS* ist sehr begrenzt - von Außen handelt es sich um eine Art WOM (Write-Only-Memory), ohne jegliche Lesefunktion. Da ich zum derzeitigen Zeitpunkt keine Verwendung für selbige Methode hatte, wurde erst einmal auf sie verzichtet. In zukünftigen Versionen wird eine Methode zum Lesen allerdings höchstwahrscheinlich noch folgen. Beim Schreiben von Zeichenketten (Funktion *nofs_writeString*) werden diese zunächst in einen Buffer zwischengelagert, der die Größe eines Datensektors auf der Speicherkarte besitzt. Um die Schreibzugriffe auf die Speicherkarte gering zu halten und trotzdem keinen großen Datenverlust zu riskieren, wird der Buffer nur dann auf die Karte geschrieben, wenn selbiger voll ist, oder es sich um den *n*-ten Funktionsaufruf handelt. Die Anzahl *n* kann in der Datei *modules/nofs.h* variiert werden (Konstante *NOFS_WRITE_BUFFER_AFTER_NTH_COMMAND*).

Die aktuelle Version der Software ist 1.1. Das AVR-Studio Projektverzeichnis kann man im Downloads-Bereich abrufen.

In Version 1.1 wurde ein Fehler behoben, durch den vorher das Schreiben immer nur bei Sektor 1 angefangen hat. Dadurch würden bereits existente Pfade allerdings überschrieben werden. Vielen Dank an Ralf Fischer für die Erkennung und gleichzeitige Lösung des Problems!

Software PC

Da beim Beschreiben der Karte auf ein Dateisystem verzichtet wurde, ist das Auslesen der Karte am PC nicht komplett ohne Zusatzsoftware möglich. Unter Linux kann man eingebaute Tools verwenden (z. B. *dd*), für Windows-PCs habe ich eine kleine Software zum Auslesen und Leeren der Karte geschrieben. Ich nenne das Tool „RawRead“. Es handelt sich hierbei um eine Kommandozeilen-Applikation, geschrieben in C++. Das Tool hat bestimmte Erkennungsmechanismen eingebaut, die das Auslesen und Leeren lediglich bei Wechselmedien erlauben. Durch einen bestimmten Header, der sich am Beginn eines NoFS-Laufwerkes befindet, wird zudem zusätzlich gewarnt, wenn man ein Nicht-NoFS-Laufwerk leeren möchte.

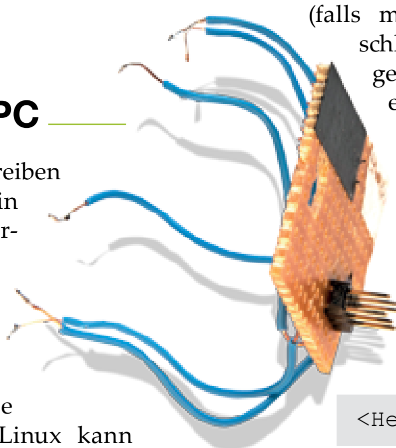


Abb. 9: ISP-Programmer

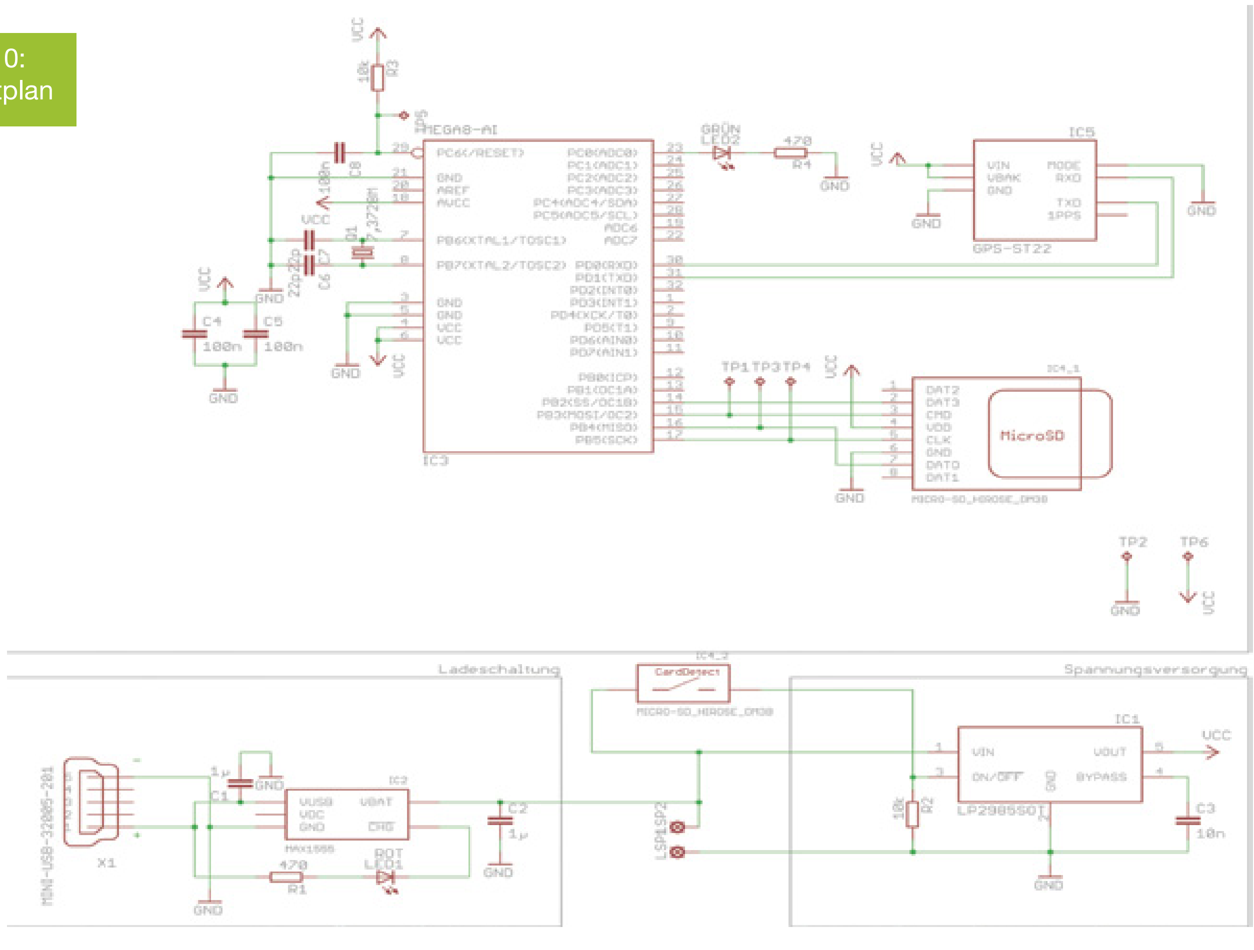
Die aktuelle Version des Programms ist 1.1. Momentan muss man Eingaben wie Zielverzeichnis oder Laufwerkswahl (falls mehrere Wechselmedien angeschlossen) zur Laufzeit tätigen, geplant ist für die nächste Version ein Akzeptieren von Parametern, wodurch man ein Wechselmedium dann mit einem Klick auslesen und leeren kann.

Das NoFS besitzt eine recht lineare Struktur und ist nach folgendem Schema aufgebaut:

```
<Header><Daten><0x03>
```

Der Header besteht aus der Zeichenfolge *k621.de* (einfallslos, ich weiß). Danach folgen fortlaufend die Daten (wobei diese kein 0x03 enthalten dürfen). Abschlusscharakter ist ein 0x03 (== ETX, **E**nd of **T**ext). Nachfolgende Daten werden ignoriert. Um eine Karte zu leeren genügt es, in den Ersten Sektor ein *k621.de<0x03>* zu schreiben. Der gLogger Mini sucht immer das erste Auftreten eines 0x03 und beginnt an dessen Stelle den Schreibvorgang. Das Dateisystem (wobei es eigentlich gar keines ist) besitzt durchaus seine Schwächen, ist jedoch für die Zwecke des GPS-Loggings komplett ausreichend und mit Mikrocontrollern

Abb. 10: Schaltplan



deutlich leichter zu Handhaben als das FAT-Dateisystem.

Das Ergebnis des Auslesevorgangs ist eine Datei, die die aufgezeichneten NMEA-Kommandos beinhaltet. Diese kann man nun beispielsweise mit meiner NMEA Toolboxzu Google-Maps Daten konvertieren. Das „RawRead“ Tool inklusive Quellcode befindet sich im Downloads-Bereich.

In der neuen Version des Programms (1.1) werden nun die vergangenen Daten nach Abfrage („Karte leeren?“) wirklich komplett gelöscht. Davor wurde lediglich der erste Sektor überschrieben. Es hat sich allerdings herausgestellt, dass es dadurch manchmal dazu kommen kann, dass beim Auslesen Teile von älteren Routen an das Ende der aktuellen Route angehängt werden. Dies ist nun nicht mehr möglich. Weiterhin bin ich von der „Dev-C++“ auf die „MS Visual C++ 2010 Express“ IDE umgestiegen, da Dev-C++ scheinbar nicht mehr ganz so gut gepflegt wird und die IDE teilweise recht fehlerbehaftet ist.

Einzelnachweise

[1] http://maps.google.com/maps/ms?ie=UTF&msa=0&msid=102721812355516657756.000489127e67aca05a27f

[2] Projektordner zum Download
http://www.mikrocontroller.net/wikifiles/9/9b/GLoggerMini_eagle.zip

[3] Änderungen im Vergleich zu den Standardfuses: CKDIV8 abgeschaltet, Clock Source = Ext. Crystal Osc., 3-8Mhz, 258/14CK + 65ms Startup Time

[4] http://www.perthold.de/BINARY/AN0003_Binary_Messages_SkyTraq_Venus6.pdf

Downloads

- AVR-Studio Projektordner V1.1p01 Bugfix
  http://www.mikrocontroller.net/wikifiles/6/61/GLoggerMini_1_1p01_avr.zip
- AVR-Studio Projektordner V1.1
  http://www.mikrocontroller.net/wikifiles/1/1a/GLoggerMini_1_1_avr.zip
- AVR-Studio Projektordner V1.0
  http://www.mikrocontroller.net/wikifiles/2/27/GLoggerMini_avr.zip
- RawRead V1.1
  http://www.mikrocontroller.net/wikifiles/b/be/GLoggerMini_rawread_1_1_exe.zip
- MS Visual C++ 2010 Express Projektordner für RawRead V1.1
  http://www.mikrocontroller.net/wikifiles/9/99/GLoggerMini_rawread_1_1_msvcpp.zip
- RawRead V1.0
  http://www.mikrocontroller.net/wikifiles/6/6b/GLoggerMini_rawread_exe.zip
- Dev-C++ Projektordner für RawRead V1.0
  http://www.mikrocontroller.net/wikifiles/1/12/GLoggerMini_rawread_devcpp.zip

EKG-Simulator mit ATTINY45

Ronny Schmiedel <ronnys30@gmx.de>

Zur Überprüfung von EKG-Geräten, EKG-Überwachungmonitoren und EKG-Langzeit Recordern benötigt man ein Simulator der Herztätigkeiten. Die kommerziell zur Verfügung stehenden Geräte sind sehr teuer und und mit mehreren zusätzlichen Funktionen ausgestattet. Ziel soll ein kleiner tragbarer EKG-Simulator sein, der einfach zu bedienen ist.



Umsetzung

Ein ATTINY45 ist hier das Herzstück des Systems. Das EKG ist digitalisiert in einem Array im Flash abgelegt und wird aller ca. 1,3msek. ausgelesen. Die EKG-Kurve wird mittels Pulsbreitenmodulation (PWM) und nachgeschaltetem 2-fach Tiefpass erzeugt. Die weiteren Spannungsteiler dienen der Aufteilung und unterschiedlichen Wichtung der Ausgänge. Damit steht an den Ausgängen das EKG-Signal als R, L, F und N zur Verfügung. Die eingebaute LED blinkt im Takt der simulierten Herzfrequenz, die 90 Schläge pro Minute beträgt. Die Stromversorgung übernimmt eine 9V-Batterie mit nachgeschalteten Längsregler 78L05. Die Betriebsdauer beträgt rein rechnerisch ca. 24h.Wird anstelle des Längsreglers ein moderner Step-Down-Regler eingesetzt ist sicher eine noch längere Betriebsdauer möglich. Zu Bedenken ist dann aber die mögliche Störung des Ausgangssignals durch den Step-Down-Regler. Von daher habe ich mich schlußendlich für den Längsregler entschieden.

Mechanischer Aufbau

Die Platine ist in ein Plastikgehäuse von Conrad-Elektronik (Best-Nr.: 522706) eingebaut (siehe Abbildung oben) und wird in das Unterteil montiert. Hierfür sind bereits die Befestiungslöcher in der Platine vorgesehen. die 9V-Batterie liegt in dem dafür vorgesehene Batteriefach. In der Frontplatte ist der Schalter für ON/OFF eingebaut und die LED. Die Kontaktierung zu den EKG-Elektroden erfolgt über selbst gedrehte Kontaktstifte. Nach dem flashen der Firmware in den Mikrocontroller mit einem Programmer über die ISP-Schnittstelle steht der Simulator zur Verfügung. Die Fuse-Bits des AVR bleiben auf Werkseinstellung von 8MHz und Teiler 8, d.h. der Systemtakt beträgt 1MHz.

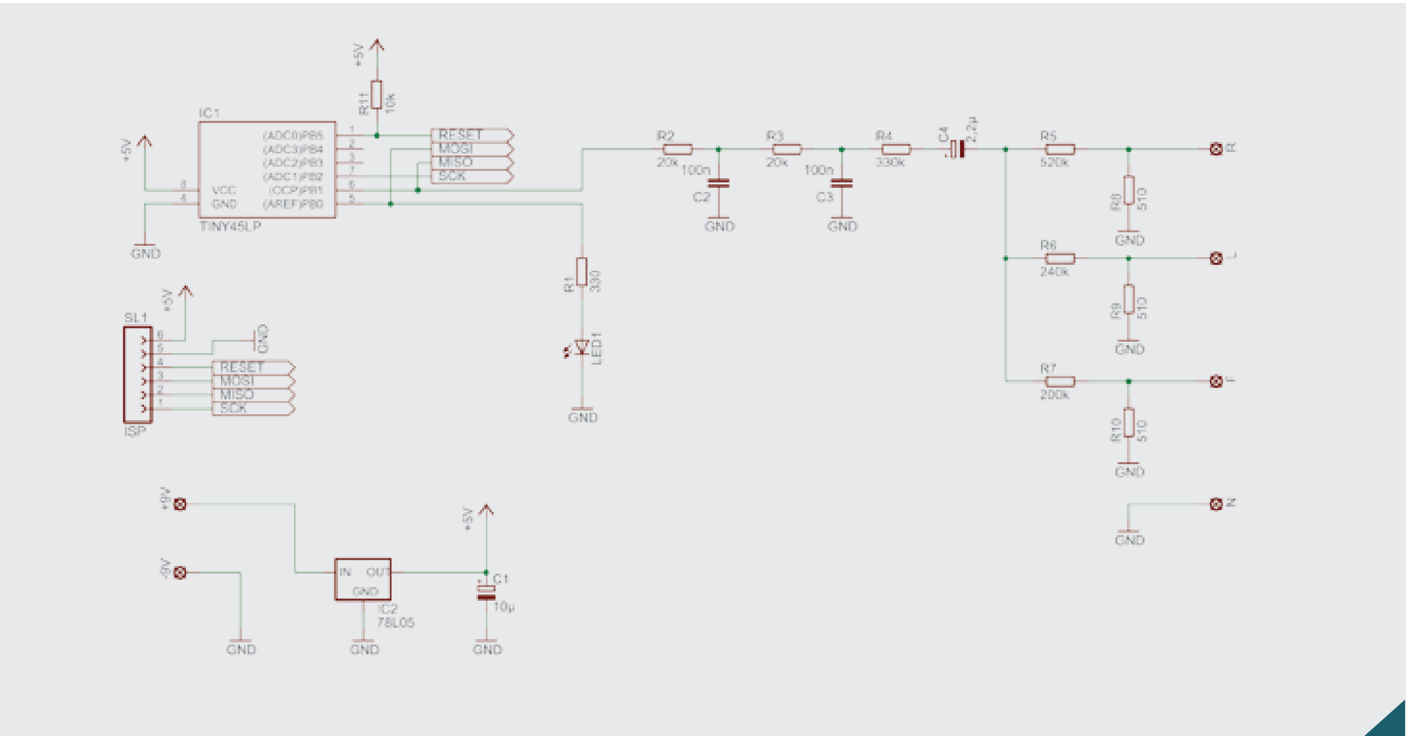


Abb. 1: Schaltplan

Software

Die Software ist in C geschrieben (mit WINAVR).

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/pgmspace.h>
#include "main.h"


//Sinus-Array
const uint8_t PROGMEM SINUS_ARRAY[118] =
{54, 60, 64, 67, 75, 75, 75, 75, 67, 64, 60, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54,
 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 30,
 30, 60, 120, 170, 171, 171, 170, 120, 43, 30, 20, 20, 25, 35, 41, 45, 49, 52, 54, 54,
 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54,
 54, 54, 54, 54, 59, 63, 66, 70, 74, 77, 80, 82, 84, 85, 85, 85, 85, 84, 82, 80, 77, 74,
 70, 65, 59, 54, 54, 54, 54, 54, 54, 54, 54, 54};

//Variablen
volatile uint8_t zeiger;                         // zum Auslesen des Arrays
volatile uint8_t zaehler = 3;                    // Zähler für Verzögerung


//Interrupt TIMER0 aller 1.3msek
ISR (TIM0_OVF_vect) {
     uint8_t data=0;
     zaehler--;
     if (zaehler == 0) {
            zaehler = 3;                         //3fach-Verzögerung Tabellenauslesens
            zeiger++;                            //zeiger erhöhen
            if (zeiger > 117){
                   zeiger = 0;                   //zeiger zurücksetzen
                   zaehler = 130;                //Nullinienverzögerung
            }
            data = (pgm_read_byte(&SINUS_ARRAY[zeiger]));
            if ((zeiger > 40) && (zeiger <= 55))      // LED einschalten
                   PORTB |= (1<<PB0);
            else
                   PORTB &= ~(1<<PB0);
            OCR0B = data;                        // Daten ins PWM Register
     }
}



int main(void) {
     DDRB |= ((1<<PB0) | (1<<PB1));      // PB0 und PB1 als Ausgang setzen
     //TIMER0 Init für PWM-EKG, PWM-Frequenz ~730Hz
     //Berechnung 1MHz/8(Prescaler)/171(CTC)= PWM-Frequenz
     OCR0A = 171;                                //CTC-Register mit 171 laden
     TCCR0A = ((1<<COM0B1)|(1<<WGM01)|(1<<WGM00));      //Clear on Compare Match, Fast PWM
     TCCR0B = ((1<<WGM02)|(1<<CS01));            //Fast PWM, Prescaler = 8
     TIMSK |= (1<<TOIE0);                        //OVF-Interrupt freischalten
     set_sleep_mode(SLEEP_MODE_IDLE);            //Init Sleep-Modus auf IDLE
     sei();
     while(1) {                                  //#### Main-loop ####
            sleep_mode();                        //AVR schlafen legen
     }
     return(0);
}
```

Ausblick

Die weitere Entwicklung geht in Richtung Miniaturisierung und Umschaltung zwischen 2 Pulsfrequenzen mittels Taster. Schlußendlich soll das Ganze mal in ein Schlüsselanhänger-Gehäuse passen mit vielleicht einem Lithium-Akku als Versorgung. Aktuelle Änderungen und Erweiterungen sind unter [1] zu finden.

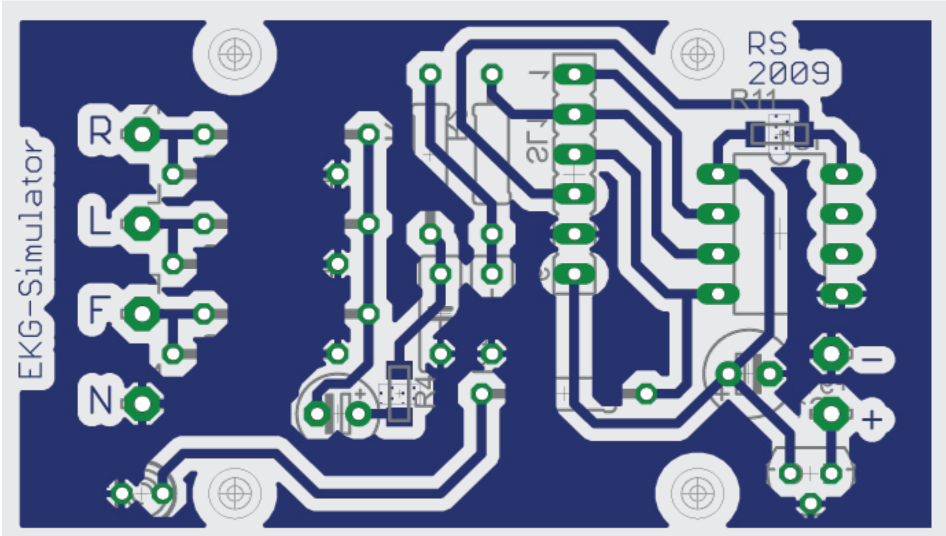


Abb. 2: Platinenlayout

Links

[1] http://www.avrs-at-leipzig.de

Touchpad - do it yourself

Dieses Platinentouchpad reagiert nicht wie die herkömmlichen Touchs auf Kapazität sondern auf den Hautwiderstand.

Tobias Frintz <Tobias-Frintz@web.de>



Auf einer Platine ist jeweils eine Minusleitung direkt neben einer Sensorleitung (siehe Abbildung 2). Beim Berühren wird die Sensorleitung hochohmig mit der Minusleitung verbunden. Bei den Pullupwiderständen gilt desto höher der Wert desto feinfühliger reagiert der Touch. Ich würde hier Werte zwischen 500 Kiloohm und 2000 Kiloohm nehmen je nachdem wie man die Touchplatine aufgebaut hat und wo und wie man sie einsetzt.

Die Auswerteeinheit

Die Auswerteeinheit ist extrem einfach: Ein 4066 CMOS-IC beinhaltet 4 digitale Schalter. Wenn die Steuerleitung an den Pluspol gelegt wird, „schaltet" der Schalter durch und wenn sie an Masse gelegt wird, ist er geöffnet. Eine Seite des Schalters ist an den Mikrocontroller angeschlossen, die andere den an Minus. Die Steuerleitungen des Schalters entsprechend an den Touch. Ein AVR wertet die Signale des 4066 aus und sendet, wenn der Touch berührt wurde, dem Hostcontroller seriell die Daten.

Touchkreis mit 4 Feldern

Eine Möglichkeit ist es, einen Touchkreis zu bauen. Natürlich kann man auch andere Formen gestalten. Bei dieser Version kann man scrollen (im Uhrzeigersinn oder gegen den Uhrzeigersinn über den Kreis fahren) und hat die Möglichkeit 4 Taster (Oben, Unten, Recht, Links) zu benutzen. Durch einen Doppelklick (wie beim Laptop) wird dem Touchcontroller gezeigt ,dass es sich um ein Tasten handelt und nicht um scrollen. Ich hab hier ein ATtiny13 eingesetzt, weil er die benötigten Portpins hat, sehr bekannt ist, oft eingesetzt wird und zudem preiswert ist.

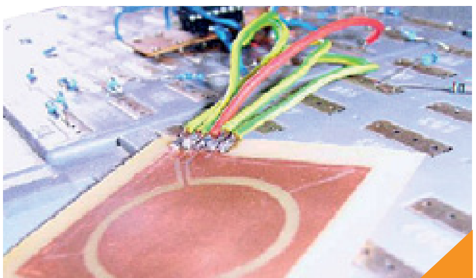


Abb. 1: Touchpad

Aufbau des Touch

Man muss bei der Platine eigentlich nur zwei Sachen beachten:

1) Der innere Kreis darf nicht direkt an den vier Sensorflächen anliegen, da sonst der Touch sehr feuchtigkeitsempfindlich wird.

2) Das Gehäuse muss die Sensorleitungen die zu den Sensorflächen führen vollständig abdecken. Man würde sonst wenn man z.B. Rechts berührt die Leitung die nach oben führt auch an Masse legen.

Abb. 2: Sensorleitung

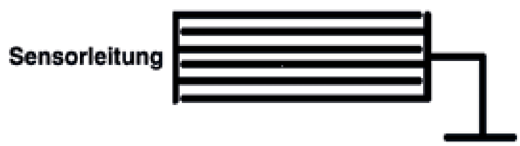
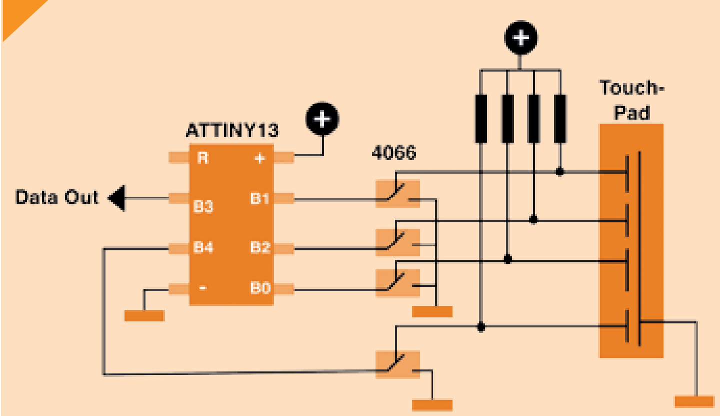


Abb. 3: Schaltbild



Funktionsweise der Software des AVRs

Im Hauptprogramm wird erst kontrolliert, ob zu hohe Feuchtigkeit herscht. Anschliessend ob die Tastensperre aktiviert werden sollte. Schließlich werden noch alle Toucheingänge abgefragt: Wurde eine Sensorfläche berührt, wird die alte Berührung mit der jetzigen verglichen. Wenn ein danebenliegendes Feld berührt wurde, wird mit dem Uhrzeigersinn scrollen oder gegen den Uhrzeigersinn scrollen in Register 18 als Zustand gespeichert. Danach oder wenn ein anderes Feld berührt wurde wird die jetzige Berührung einfach in das Register, wo die alte Berührung ist, entsprechend verschoben.

Hinweis: Dieses Programm ist lediglich ein Testprogramm, welches die Berührungen über 3 Leds anzeigt. Wenn man den Touch in der Praxis einsetzen will, muss man das Programm umschreiben und eventuell Funktionen wie: Tasten hinzufügen , etc. erweitern. Integriert werden kann die Software entweder über einen Timerinteruppt (im Host) oder wie oben beschrieben mittels z.B externen ATtiny, welcher über eine Schnittstelle Daten mit dem Host übermitteln kann.

```
;+----------------------------------------------------------------------
;| Title                : Touchkreiscode
;+----------------------------------------------------------------------

;| Schaltung    : Die Anschlüsse PB4, PB2, PB1, PB0 sind an die 4
;|                4066 IC-Ausgänge angeschlossen. Die Anderen an
;|                Minus. Die Steuereingänge des 4066 sind an die
;|                4 Touchfelder angeschlossen mit hochohmigen
;|                Widerständen zu Plus (damit die Eingänge nicht
;|                wahllos an und ausschalten)
;+----------------------------------------------------------------------
;| Prozessor    : ATtiny13 , ATmega8
;| Takt         : egal
;| Sprache      : Assembler
;| Datum        : heute
;| Version      : 1.1
;| Autor        : Tobias Frintz, 15
;|                Tobias-Frintz@web.de
;+----------------------------------------------------------------------
.include        "AVR.H"
;-----------------------------------------------------------------------
;Reset and Interrupt vector             ;VNr.  Beschreibung
        rjmp    main    ;1    POWER ON RESET
        reti            ;2    Int0-Interrupt
        reti            ;3    Int1-Interrupt
        reti            ;4    TC2 Compare Match
        reti            ;5    TC2 Overflow
        reti            ;6    TC1 Capture
        reti            ;7    TC1 Compare Match A
        reti            ;8    TC1 Compare Match B
        reti            ;9    TC1 Overflow
        reti            ;10   TC0 Overflow
        reti            ;11   SPI, STC Serial Transfer Complete
        reti            ;12   UART Rx Complete
        reti            ;13   UART Data Register Empty
        reti            ;14   UART Tx Complete
        reti            ;15   ADC Conversion Complete
        reti            ;16   EEPROM Ready
        reti            ;17   Analog Comparator
        reti            ;18   TWI (I²C) Serial Interface
        reti            ;19   Store Program Memory Ready
; r16 ist das Arbeitsregister
;
; r17 speichert die alte Berührung:
;      oben   = 1
;      rechts = 2
;      unten  = 3
;      links  = 4
;
; r18 ist das Register in dem das Ausgangssignal steht
```

```
;       Feuchtigkeitsfehler     = 0b00000001
;       Tastensperre            = 0b00000111
;       Gegen den Uhrzeigersinn = 0b00000010
;       Mit dem Uhrzeigersinn   = 0b00000100
;       Obengetastet            = 0b00000101
;----------------------------------------------------------------------
;Start, Power ON, Reset
main:   ldi     r16,lo8(RAMEND)
        out     SPL,r16
        ldi     r16,hi8(RAMEND)
        out     SPH,r16
        sbi     DDRC,0  ;Leds auf Output
        sbi     DDRC,1
        sbi     DDRC,2
        sbi     PORTB,0 ;Pullup einschalten für Touch
        sbi     PORTB,1
        sbi     PORTB,2
        sbi     PORTB,4
        ldi     r18,0b00000000
        clr     r19
        clr     r20
        clr     r21
        clr     r22
;----------------------------------------------------------------------
mainloop:       wdr
        rcall   Tastensperre
        out     PORTC,r18
        sbis    PINB,0
        rjmp    Obenberuehrt
        sbis    PINB,1
        rjmp    Linksberuehrt
        sbis    PINB,2
        rjmp    Untenberuehrt
        sbis    PINB,4
        rjmp    Rechtsberuehrt
        rjmp    mainloop
;----------------------------------------------------------------------
Obenberuehrt:
        ldi     r16,2
        cp      r17,r16
        breq    im _ uhrzeigersinn
        ldi     r16,4
        cp      r17,r16
        breq    geg _ uhrzeigersinn
        ldi     r17,1
        rjmp    mainloop
;------------------------------------------------------------------------
Rechtsberuehrt:
        ldi     r16,3
        cp      r17,r16
        breq    im _ uhrzeigersinn
        ldi     r16,1
        cp      r17,r16
        breq    geg _ uhrzeigersinn
        ldi     r17,2
        rjmp    mainloop
;-----------------------------------------------------------------------
Untenberuehrt:
        ldi     r16,4
        cp      r17,r16
        breq    im _ uhrzeigersinn
        ldi     r16,2
        cp      r17,r16
        breq    geg _ uhrzeigersinn
        ldi     r17,3
        rjmp    mainloop
```

```
;-------------------------------------------------------------------
Linksberuehrt:
        ldi     r16,1
        cp      r17,r16
        breq    im _ uhrzeigersinn
        ldi     r16,3
        cp      r17,r16
        breq    geg _ uhrzeigersinn
        ldi     r17,4
        rjmp    mainloop
;-------------------------------------------------------------------
 Tastensperre:
        sbic    PINB,0
        ret
        sbis    PINB,1
       ret
        sbic    PINB,2
        ret
        sbis    PINB,4
        ret
        ;jetzt ist die tasstensperre aktiviert
        ldi     r18,0b00000111
        out     PORTC,r18
        ldi     r17,100

Aktiviert:      wdr
        rcall   Feuchtigkeit
        sbis    PINB,0
        rjmp    Aktiviert
        sbic    PINB,1
        rjmp    Aktiviert
        sbis    PINB,2
        rjmp    Aktiviert
        sbic    PINB,4
        rjmp    Aktiviert
        ldi     r18,0b00000000
        out     PORTC,r18
        ret
;-------------------------------------------------------------------
im _ uhrzeigersinn:
        ldi     r18,0b00000100
        ldi     r17,100
        rjmp    mainloop
;-------------------------------------------------------------------
geg _ uhrzeigersinn:
        ldi     r18,0b00000010
        ldi     r17,100
        rjmp    mainloop
;-------------------------------------------------------------------
 Feuchtigkeit:
       sbic    PINB,0    ; Wenn auf dem Touchpad Wasser ist
        ret               ; sind alle Touchfelder "berührt"
        sbic    PINB,4    ; dieses Unterprogramm ist dazu da
        ret               ; diesen Fehler zu erkennen und es
        sbic    PINB,2    ; dem Host mitzuteilen
        ret               ; "Reinigen sie ihr Touchfeld"
        sbic    PINB,1
        ret
        rjmp    Feuchtigkeitsfehler
;-------------------------------------------------------------------
Feuchtigkeitsfehler:
        ldi     r18,0b00000001
        out     PORTC,r18
        ldi     r17,100
        rjmp    Feuchtigkeitsfehler
```

Touchkreis mit mehr als 4 Feldern

Wenn man ein Touchkreis mit mehr als 4 Touchfelder bauen will, braucht man für jede weiteren 4 Felder zusätzliche einen 4066 IC. Die ist eine platzraubenende und teuere Lösung. Deswegen habe ich ein Touchkreis entwickelt, mit dem man beliebig viele Felder (Anzahl durch 3 teilbar sind) mit nur einem IC auswerten kann. Zusätzlich kann man wie bei dem einfacheren 4-Feld-Touchkreis Tasten einzeln auswerten. Jedoch kann man hier nur 3 Tastfelder (mit Dioden) auswerten ohne noch einen Zusätzlichen IC zu verwenden. Das Prinzip von diesem Touch ist sehr einfach:

Wenn ein Berührung von ...

- 1 nach 2 stattfand ist einmal mit dem Uhrzeigersinn
- 2 nach 3 stattfand ist einmal mit dem Uhrzeigersinn
- 3 nach 1 stattfand ist einmal mit dem Uhrzeigersinn
- 1 nach 3 stattfand ist einmal gegen den Uhrzeigersinn
- 3 nach 2 stattfand ist einmal gegen den Uhrzeigersinn
- 2 nach 1 stattfand ist einmal gegen den Uhrzeigersinn

... gescrollt worden.

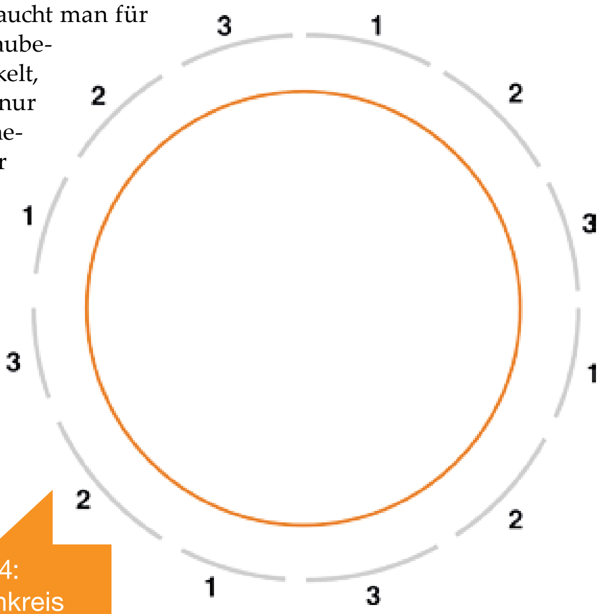


Abb. 4: Touchkreis beliebig

Tasten

Wenn man auf dem Touchpad auch „OK“ und „Abrechen“ oder ähnliche „Taster“ zusätzlich zu dem scrollen integrieren könnte, würde man kein einzigen echten Taster mehr benötigen und hätte ein extrem schickes robustes Design (wenn man will sogar Wasserdicht). Hier die Idee zur Umsetzung:

Tasten bei dem Touch mit 4 Feldern:

```
mainloop:    wdr
     rcall   Tastensperre
     out     PORTC,r18
     sbis    PINB,0
     rjmp    Obenberuehrt
     sbis    PINB,1
     rjmp    Linksberuehrt
     sbis    PINB,2
     rjmp    Untenberuehrt
     sbis    PINB,4
     rjmp    Rechtsberuehrt
     rjmp    Nichtsberuert
     rjmp    mainloop
```

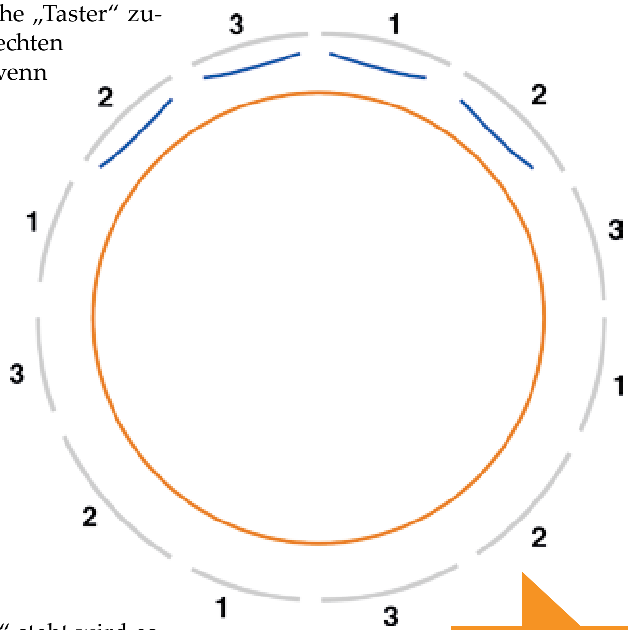


Abb. 5: Touchkreis 2

Wenn man am Ende der mainloop das Unterprogramm „Nichtsberuert“ steht wird es immer aufgerufen falls kein Feld berührt wurde. Inkrementiert man jedesmal den Inhalt eines Registers sobald das Unterprogramm aufgerufen wurde, lässt sich feststellen wie lange der Zustand „Nichtsberührt“ angehalten hat. Wird dann z. B. oben berührt und war die alte Berührung auch oben muss man das „Nichtsberührtregister“ überprüfen. Ich würd sagen zwischen 0,5 und 2 Sec kann man das Tasten zählen lassen.

Tasten bei dem Touch mit mehr als 4 Feldern:

Das Prinzip ist genau das gleiche wie bei dem Touch mit 4 Feldern. Nur bei diesem hier kann man nicht die 3 sich immerwiederholenden Felder als Taster nehmen weil sie an verschiedenen Stellen am Touch vorkommen und man so nicht feststellen kann, wo das Touch berührt wurde.Zum Glück hat man ja noch ein 4066 Eingang frei. Wenn man 4 Dioden an 4 nebeneinanderliegende Felder anschließt hat man schon ein Tastfeld in dem Kreis. Man scrollt dann durch das Menü bestätigt mit einem Doppelklick auf dem Touch oben und als letzten Menüpunkt würde ich Zurück oder Abbrechen nehmen.

Probleme

Bei zu kleinen Touchfeldern:

Man muss darauf achten das man die Touchfelder (vorallem bei dem Touchkreis mit mehr als 4 Feldern) nicht zu klein gestaltet sind, denn sonst berührt man beim darüberfahren vielleicht zufällig ein Feld nicht und dann kommt es zu Fehlern beim Auswerten.

In Standgeräten:

Das Problem in Standgeräten ist, dass man normalerweise waagerecht tippt. Daher berührt nur die Fingerkuppe den Touch. Ich würde hier die Sensorflächen ein bisschen näher an den Minuskreis legen, außerdem die Pullupwiderstände vor den 4066er Eingängen erhöhen (nicht zu hoch sonst treten Probleme auf wenn man schwitzt, dagegen Haucht oder sich in stark feuchten Räumen befindet).

In Handgeräten:

Hier hat man den Vorteil, dass man fast parallel zum Gehäuse den Touch berührt (-> große Auflegefläche daher kann man die Pullups vor den 4066er Eingängen ein bisschen senken) In Handgehäusen bietet sich oft nicht viel Platz. Daher würde ich eine kleine doppelseitige Platine verwenden auf der eine Seite der Touch und auf der Rückseite die Auswerteelektronik komplett in SMD kommt.

Pressemitteilungen

Jetzt als mobile App für iPhone und iPod Touch: Die PCB-Toolbox für Leiterplattenentwickler



Nach großer Resonanz zur beliebten Toolbox auf www.leiton.de hat die LeitOn GmbH eine mobile App erstellt. Für nur 0,79 Euro ist diese App mit 6 verschiedenen Tools bei iTunes erhältlich und beinhaltet Berechnungshilfen, die sonst erheblich teurer angeboten werden. Hier finden Entwickler, die mit Wärmemanagement von Aluminiumleiterplatten, Hochfrequenzen, Hochstromanwendungen und flexiblen Leiterplatten zu tun haben nützliche Werkzeuge zur Berechnung kritischer Werte, teils mit berechneten Kennlinien. Hier ein kurzer Überblick der enthaltenen Tools:

- Wärmemanagement mit Aluminiumleiterplatten
- Leiterbahnerwärmung über Strom
- Biegeradien von flexiblen Leiterplatten
- Spannungsabfall auf Leiterbahnen
- Impedanzbetrachtung
- Volumen- Gewichtsrechner
- Mobil, online oder persönlich: LeitOn bietet Ihnen umfassende Beratung und Service

Leiton: http://www.leiton.de

Pressemitteilungen im embedded projects Journal

Im embedded projects Journal sollen ab sofort regelmäßig Pressemeldungen gedruckt werden. Primär richtet sich das Angebot an alle Beteiligten Leser und Sponsoren des Hefts. Neue Produkte, Interessante Links, etc. können einfach und unkompliziert präsentiert werden.

Für Fragen und Pressemeldungenwendet euch bitte an info@embedded-projects.net. Inhalte von Pressemitteilungen könnten sein:

- Neue Produkte aus dem Bereich Embedded Systeme
- Wissenwertes aus der Branchen
- Spannende kleine Open-Source Projekte
- Ankündigungen für Linux Tag bzw. Hardwaretreffen in einer Region oder Stadt

Kontakt: info@embedded-projects.net

Die neue Homepage ist online:



http://journal.embedded-projects.net

Das embedded projects Journal im Papierabo für 12,99 EUR pro Jahr

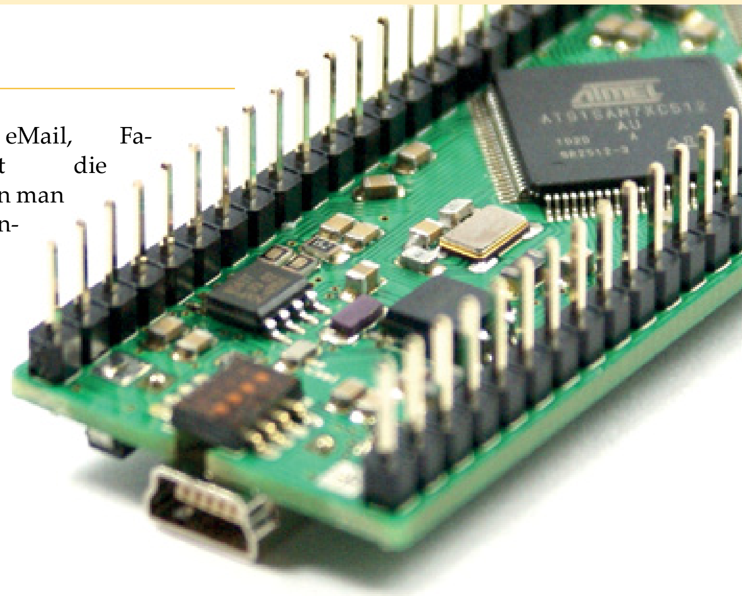
Ab sofort können Leser das Open-Source Projekt auch finanziell unterstützen. Wie? Ganz einfach. Man kauft sich online über die Homepage http://journal.embedded-projects.net das Jahresabo für die Papierversion. Durch diese Einnahmen können Druck und Porto für die Papierversion finanziert werden. Zu dem gibt es immer mehr sogenannte Postboten, dass sind Lehrer, Professoren oder Schüler/Studenten die die Ausgabe in Stapeln von 5-20 in den Lehreinrichtungen auslegen.

Für Fragen rund um das Abo wendet euch bitte an info@embedded-projects.net.

Web-basiertes Messen, Steuern, Regeln mit dem eNet-sam7X

Mikrocontroller Modul mit OpenSource Multitasking Betriebssystem

Ole Reinhardt <ole.reinhardt@embedded-it.de>



Einleitung

Wer glaubt das Internet besteht nur aus eMail, Facebook und YouTube etc, der verpasst die aufregende Welt der embedded Ethernet Geräte. Wie wäre es, wenn man seine Geräte von überall auf der Welt kontrollieren kann? Die Solaranlage mit dem Smart-Phone vom Strand aus steuern, die Heizung zum Feierabend vom Büro aus einschalten und die Rolläden automatisch steuern? Aber nicht nur für das Eigenheim, natürlich auch für die Industrie ist dies von großer Bedeutung. So kann man seine Produktionsprozesse von jedem Ort der Welt ganz einfach über das Internet kontrollieren. Vorausgesetzt die Steuerung hat einen Zugang zum weltweiten Netz.

Und genau darum geht es hier...

Im Folgenden zeigen wir Schritt für Schritt, wie man mit geringem Aufwand innerhalb kürzester Zeit zu einem vollständigen embedded Webse verkommt, der einfache Steuerungsaufgaben übernehmen kann. Das Projekt ist absichtlich sehr einfach gehalten und soll nur das Prinzip beschreiben. Deutlich komplexere Systeme sind jederzeit problemlos möglich.

Die Projektanforderungen

Konkret wollen wir einen kleinen embedded Webserver mit folgenden Anforderungen bauen:

- Wenig externe Bauteile
- Steuerung von 2 Relais über ein Web-Interface
- Webseiten sollen auf Micro-SD Karte gespeichert werden und einfach austauschbar sein
- Steuerung der Relais über ein einfaches CGI Interface
- Fixe Netzwerkkonfiguration (IP Adresse)

Das Ziel ist ein einfacher „Internet-Schalter". Also ein Gerät, mit dem man über eine Webseite z.B. zwei Geräte ein und aus schalten können soll.

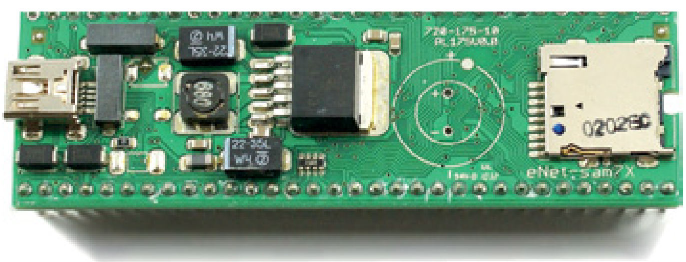


Abb. 1: eNet-sam7X

ARM7 Webserver Kompakt

Schon wieder ein neues Mikrocontroller Modul werden sich die einen Leser denken, das ist genau die Lösung, nach der ich schon immer gesucht habe, die anderen. Die Rede ist von einem neuen embedded Modul auf Basis einer ARM7 CPU von Atmel, welches viel Leistung und viele I/O Schnittstellen auf kleinstem Raum bietet und sich ideal in eigene Applikation einbinden lässt.

Die Idee für die Entwicklung des Moduls ergab sich aus der Suche nach einer kompakten Steuerung, die sich einfach in C programmieren lässt, einen Ethernet Port besitzt und in der Lage ist auch komplexe Peripherie zu steuern sowie Messdaten direkt zu verarbeiten und speichern.

Die zweite wichtige Voraussetzung war, dass das Modul vollständig durch OpenSource Software unterstützt werden muss. Ein echtes Multitasking Betriebssystem mit vollständigem TCP/IP Stack sollte zum Einsatz kommen.

So war die Idee geboren. Ein kompaktes Mikrocontroller Modul auf Basis des AT91SAM7XC512 von Atmel wurde entwickelt und ein Board Support Package auf Basis des OpenSource Echtzeit Betriebssystems (RTOS) Nut/OS wurde zusammen gestellt.

eNet-sam7X

Im Herzen ganz groß, im Platz ganz klein.

Das Herz des Boards ist eine ARM7 CPU von Atmel, der AT91SAM7XC512. Dieser Mikrocontroller bietet alles, was das Entwickler Herz höher schlagen lässt. Neben 512 KB internem Flash für eigene Programme und Daten sowie 128KB statischem RAM bietet er diverse Peripheriekomponenten wie z.B. 100 MBit Ethernet, zwei vollwertige und eine Debug UART, einen CAN-Bus, zwei SPI- und einen I2C Bus, ein I2S Interface, Timer, GPIOs, PWMs, A/D Wandler USB 2.0 und vieles mehr. Ergänzt wird der Mikrocontroller auf dem Board durch einen Ethernet PHY von Davicom, ein 4 MB Dataflash (ideal als großer Datenspeicher für logging Aufgaben, Webseiten, User Daten usw.), einer Batterie gepufferten Echtzeituhr (RTC) sowie einem 3.3V Schaltregler, einem Mini-USB Connector sowie einem Micro-SD Slot (Noch mehr Speicher für große Datenmengen). Und das alles auf der Fläche eines DIL64 ICs (siehe Abbildung 1 und Titelbild).

Viele Mikrocontroller Boards und Evaluations-Plattformen sind als eigenständiges Gerät konzipiert und schränken die Nutzbarkeit damit auf den gegebenen Form-Faktor sowie die vorhandene Peripherie ein. Das eNet-sam7X nicht! Es ist explizit als Drop-In Lösung zur Entwicklung eigener Steuerungssysteme und Web-Appliances gedacht und bietet dem Entwickler daher auf seinen zwei Steckerleisten alle freien Signale der SAM7 CPU, das Ethernet Interface, USB, MMC, JTAG sowie einen Spannungsausgang mit 3.3V, der zur Versorgung des Baseboards verwendet werden kann. Da das Board einen eigenen Schaltregler besitzt kann so die gesamte Schaltung einfach eine eine externe 5-24V Spannungsquelle angeschlossen oder z.B. über den USB Bus mit Strom versorgt werden.

Der Schaltplan des eNet-sam7X ist unter [1] zu finden.

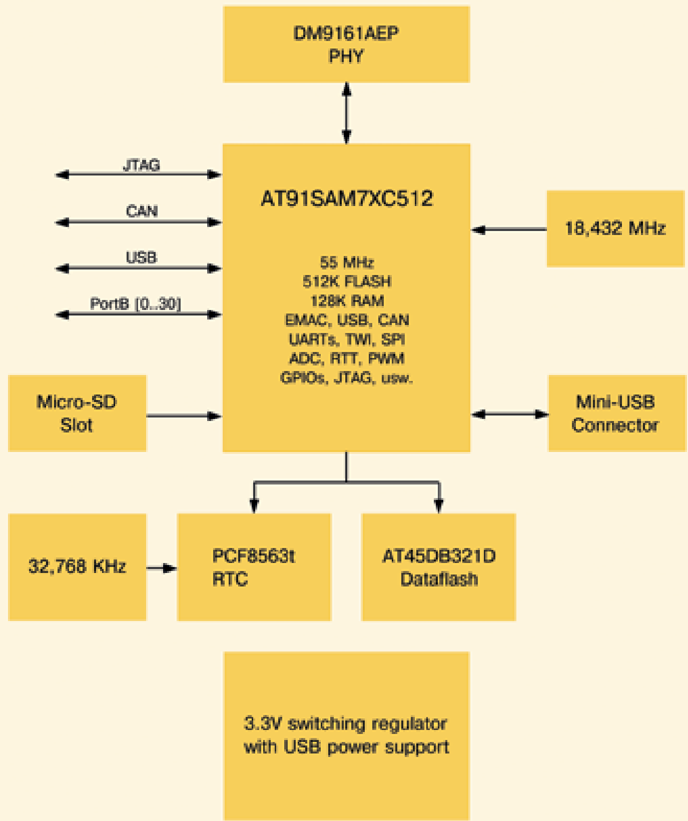


Abb. 2: Blockdiagramm

Die Schaltung

Wie einfach es ist, mit dem eNet-sam7X Modul zum Beispiel einen „Internet-Schalter“ zu bauen, zeigt Abbildung 3.

Achtung: Zu beachten ist, dass diese Schaltung lediglich ein Schaltungsbeispiel ist und bei der Verwendung von Netzspannung in eigenen Schaltungen diverse Vorschriften zu beachten sind, auf die an dieser Stelle nicht näher eingegangen wird.

Das Herz der Schaltung ist natürlich das eNet-sam7X Modul. Als externe Beschaltung ergibt sich lediglich noch die Ethernet Buchse (hier eine mit Auto MDIX Funktionalität, d.h. automatischer Kreuzung des Ethernet Kabels) mit den Terminierungs-Widerständen, ein Reset-Taster, zwei Relais mit Ansteuertransistoren sowie ein JTAG Stecker (10-polige ARM Belegung) und eine Stiftleiste um zu debug Zwecken auf die RX / TX Signale der UART0 zugreifen zu können.

Die Belegung dieser Stiftleiste erlaubt direkt das Aufstecken des Sparkfun „FTDI Basic Breakout – 3.3V“ Moduls, einem einfachen 3.3V RS232 => USB Converter. Die Schaltung kann direkt mit 12V – 24V betrieben werden, je nach der Betriebsspannung der gewählten Relais. Die 3.3V Versorgungsspannung für das Mikrocontroller Modul wird direkt auf dem Modul mit einem Schaltregler erzeugt. Das Modul stellt die 3.3V auf Pin 34 zur Verfügung, so dass weitere Hardware auf dem Baseboard gleich mit versorgt werden könnte.

Die beiden Relais sind über Ansteuertransistoren an die GPIO Pins PA21 und PA22 des AT91SAM7XC512 angeschlossen. Die Transistoren werden benötigt, da die CPU nicht genügend Strom liefern kann, um die Relais direkt zu schalten. Die Relais sind in diesem Beispiel als Wechsler ausgelegt und können über die Ausgänge an Klemme KL2 je einen Verbraucher (z.B. eine Lampe) schalten. Im Folgenden möchte ich zeigen, wie einfach es ist diese Relais über eine Webseite zu steuern.

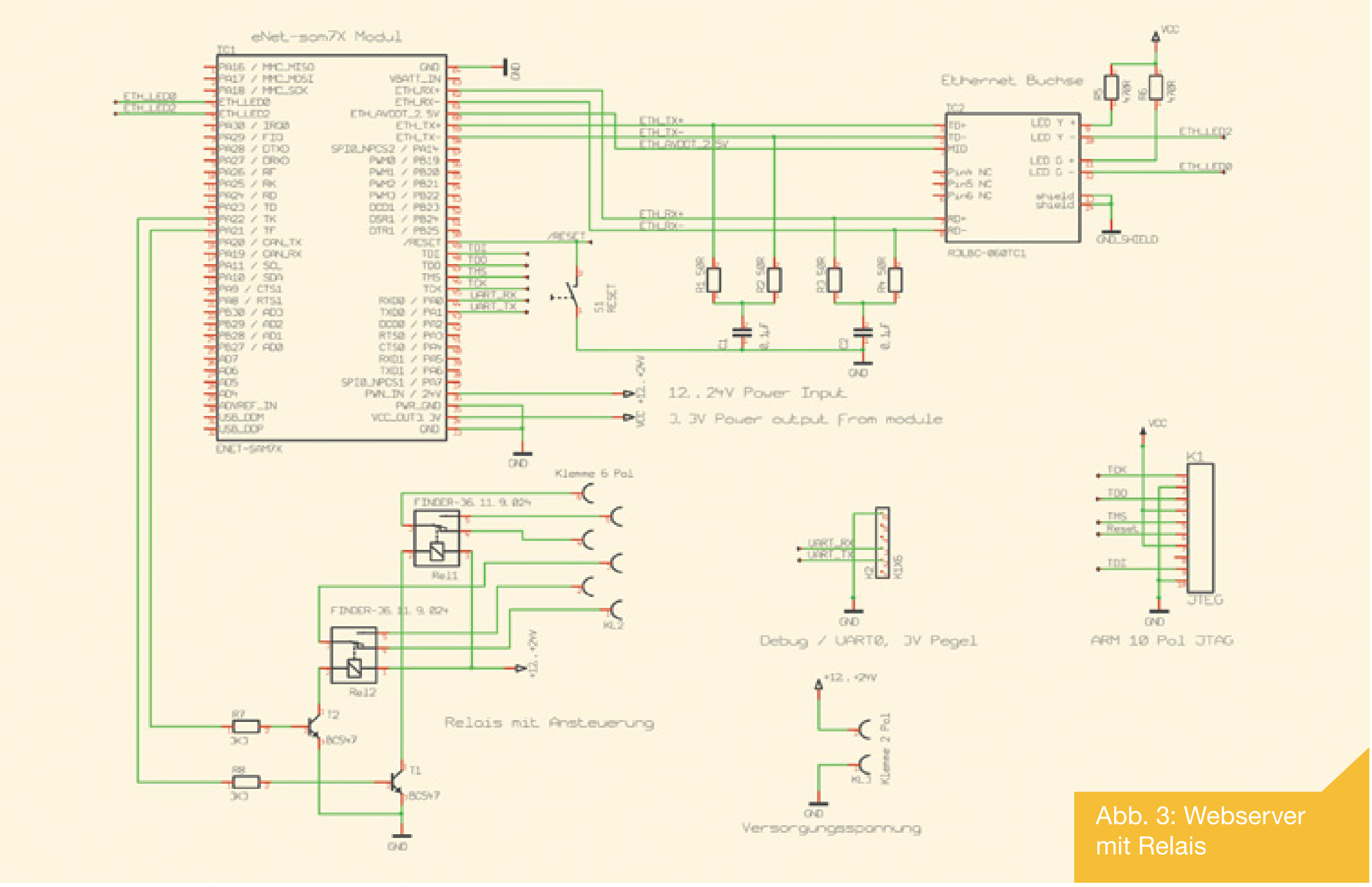


Abb. 3: Webserver mit Relais

Nut/OS

Ein modulares OpenSource Echtzeit-Betriebsystem mit TCP/IP Stack für Mikrocontroller

Eine komplexe Hardware wie diese ganz ohne Betriebssystem programmieren? Das ist aufwändig, kostet viel Zeit für die Einarbeitung und bietet lauter Fehlerquellen. Warum also nicht nach einer Lösung suchen, die uns das Entwickeln von Treibern weitestgehend abnimmt und ein Programmiermodell bietet, wie wir es vom PC her kennen? Das soll gehen? Auf einem Mikrocontroller?

Gesucht, gefunden! Das Ethernut Projekt [2] mit seinem Open-Source Echtzeit-Betriebssystem Nut/OS [3] ist genau die richtige Lösung. Das Ethernut Projekt wurde 2001 gegründet und seit dem wurde das Nut/OS zu einem sehr modularen und leicht konfigurierbaren Multitasking-Betriebssystem für Mikrocontroller entwickelt. Ursprünglich für 8 Bit Architekturen (in erster Linie Atmel AVR) entwickelt, unterstützt es heute eine ganze Reihe unterschiedlicher CPUs. Unter anderem den hier verwendeten AT91SAM7XC512 von Atmel. Nut/OS steht unter der BSD Lizenz und ist damit ohne Probleme für Closed- wie auch für OpenSource Projekte verwendbar.

Das Betriebssystem skaliert hervorragend mit den Möglichkeiten der jeweiligen Architektur. Durch den modularen Aufbau des Betriebssystem werden nur die Teile in eigene Programme eingebunden, die von der Anwendung tatsächlich benötigt werden. Das hält den Code klein und spart Ressourcen, von denen man auf einem Mikrocontroller bekanntlich nie genug hat.

Die Anpassung an das Zielsystem erfolgt in der Regel automatisch. Zur Feinabstimmung steht mit dem Nut/OS Konfigurator eine grafische Oberfläche unter Linux, Windows und OS X zur Verfügung.

Nut/OS bietet echtes Multitasking und ist dabei sicher und einfach anzuwenden. Sein kooperatives Multitasking garantiert, dass ein Thread die Kontrolle nur an eindeutig definierten Punkten abgibt. In den meisten Fällen kann so ohne zusätzliche Absicherung auf gemeinsam genutzte Ressourcen zugegriffen werden. Daraus ergibt sich kleiner und einfacher Code für die Anwendung, sowie ein geringes Risiko von Race-Conditions und Deadlocks. Deterministische Interrupt Latenzzeiten bieten hartes Echtzeitverhalten innerhalb festgelegter Zeitgrenzen, unabhängig von aktuell verfügbaren Ressourcen.

Aber das Beste zum Schluss, Nut/OS bring mit Nut/NET einen vollständigen TCP/IP Stack mit BSD artigem Socket Interface mit. Neben TCP und UDP Sockets sind diverse High-Level Protokolle implementiert. z.B. ARP, ICMP, PPP, DHCP, DNS, SNTP, FTP, TFTP, SYSLOG, HTTP, WINS (Subset) usw.

Das gesamte API Design von Nut/OS ist an den POSIX Standard angelehnt und verwendet bekannte Treiberkonzepte wie z.B. Character- und Block-Devices, Bus-Abstraktionen und dergleichen mehr. Das macht den Einstieg einfach und mittels der diversen Beispiele findet sich auch der Einsteiger schnell zurecht

Erste Schritte: Die Vorbereitungen

Damit wir loslegen können, braucht es natürlich noch einen Cross-Compiler bzw. eine Toolchain für das Board. Das Nut/OS ist auf die Verwendung der GNU ARM Compiler Suite ausgelegt. Diese gibt es sowohl für Windows als auch für Linux und den Mac. Für Windows und Linux bietet sich zum Beispiel die eCross ARM Toolchain [4] oder die Yagarto Toolchain [5] an. Beide sind OpenSource und natürlich frei verfügbar. Sie enthalten den GNU C Compiler, Debugger sowie die Standard C Library

(Newlib). Um an dieser Stelle nicht auszuufern, gehe ich nicht weiter auf die Installation ein und möchte auf unsere Webseite [6] verweisen.

Neben der Toolchain benötigen wir jetzt natürlich noch das Nut/OS Betriebssystem. Ab der Ethernut Version 4.9.11 wird die Unterstützung für das eNet-sam7X Board in der offiziellen Ethernut Distribution enthalten sein. Bis dahin kann man sich ein angepasstes Ethernut Paket auf der Webseite zum eNet-sam7X Board Support Package [6] herunter laden. Hier findet sich auch eine entsprechende Installationsanleitung.

Jetzt kann es losgehen: Der eigene Webserver

Alles ist soweit vorbereitet, jetzt kann es losgehen! Der eigene Webserver soll erstellt werden. Was auf den ersten Blick furchtbar kompliziert klingen mag, ist den zweiten Blick eigentlich ganz einfach, da das meiste bereits von den Nut/OS Bibliotheken abgenommen wird.

Unser Programm muss dafür mehrere Dinge tun:

1) Hardware initialisieren (GPIO Pins Konfigurieren, UART Treiber laden für Debug-Zwecke, Netzwerktreiber initialisieren, MMC Treiber initialisieren.

2) Micro-SD Karte mounten (die MMC Karte dient uns als Speicherort für die Webseiten)

3) Netzwerk konfigurieren (IP Adresse einstellen)

4) CGI Funktionen registrieren und Webserver Threads starten.

5) Auf eingehende Verbindungen von einem Webbrowser warten und diese bearbeiten. Relais setzen, wenn die korrekte Anforderung vom Webbrowser geschickt wurde.

All das passt in ca. 250 Zeilen Code! Das meiste erledigen die Nut/OS Webserver Funktionen bereits im Hintergrund. Der Übersichtlichkeit halber möchte ich an dieser Stelle den Programmcode nur ausschnittsweise besprechen. Der vollständige Sourcecode kann unter [7] heruntergeladen werden.

Zunächst wollen jedoch kurz das Benutzer Interface unseres „Internet-Schalters“ vorstellen...

Die Webseite

Da wir ja ein embedded Gerät bauen, hat unsere Anwendung bzw. unser Gerät natürlich kein klassisches grafisches Benutzerinterface. Es ist ja gerade unsere Intention ein Gerät zu bauen, welches von überall auf der Welt gesteuert werden kann. Also verlagern wir unser Benutzerinterface einfach auf eine Webseite. Auf dieser Webseite können wir natürlich wie auf einer klassischen Webseite diverse grafische Gestaltungselemente, Info- und Hilfs-Texte unterbringen und zusätzliche Informationen ablegen. Das eigentlich Interessante jedoch ist, dass wir Betriebszustände unseres Gerätes, Messwerte und diverses mehr darstellen können. Hierzu eignen sich hervorragend die Web-2.0 Technologien. Ein Schlagwort „Ajax“ ist sicher jedem ein Begriff. An dieser Stelle wollen wir jedoch bei einem einfachen Beispiel bleiben und über einfache Web-Formulare mit unserer Anwendung kommunizieren. Wer jedoch selber mit Ajax und anderen Web-2.0 Technologien experimentieren möchte, dem kann ich an dieser Stelle nur die diversen freien und kommerziellen Javascript Bibliotheken wie z.B. JQuery, ExtJS etc. empfehlen. Ein nettes Beispiel für eine ähnliche Anwendung mit dynamischer Ajax Unterstützung findet man unter [9]. Es handelt sich dabei um die eNet-sam7X basierte Solaranlagen Steuerung meines Kollegen.

Die Webseite für unseren Webserver könnte wie in Abbildung 4 dargestellt aussehen.



Abb. 4: Webseite

Der eigentliche funktionale Kern der Seite ist hier ein einfaches Formular. Dir 4 „Set“ und „Clear“ Buttons (je einer pro Relais) sind dabei als einfache Submit-Buttons mit einem eindeutigen Namen (set 1, clear1, set2, clear2) markiert.

Der HTML-Code für das Formular sieht dabei wie folgt aus:

```
<table bgcolor="#E0E0E0" height="130" width="300">
            <tr>
                   <TD height="29" width="100">Relais 1:</TD>
                   <TD height="29" width="100"><input type = „submit" name="set1" value="Set"
                   size="8"></TD>
                   <TD height="29" width="100"><input type = „submit" name="clear1"
                   value="Clear" size="8"></TD>
            </tr>
            <tr>
                   <TD height="29" width="100">Relais 2:</TD>
                   <TD height="29" width="100"><input type = „submit" name="set2" value="Set"
                   size="8"></TD>
                   <TD height="29" width="100"><input type = „submit" name="clear2"
                   value="Clear" size="8"></TD>
            </tr>
     </table>

</form>
```

Klickt der Benutzer nun z.B. auf den Button „Set“ in der Zeile Relais 1, so sendet der Webbrowser einen HTTP Post Request an unseren Webserver und überträgt dabei den Namen des Buttons an die CGI Funktion („cgi-bin/relais.cgi“). Diese wertet die übertragenen Parameter aus und schaltet anhand der Button-Namen das entsprechende Relais entweder ein oder aus. So einfach lässt sich ein „Internet-Schalter“ realisieren. Die oben abgebildete Webseite liegt unserem Beispiel natürlich bei.

Das Programm

Nachdem wir jetzt bereits unser Benutzerinterface (die Webseite) vorgestellt haben, fehlt noch das eigentliche Programm, welches wir auf das eNet-sam7X Board laden und dort ausführen können.

Fangen wir also zunächst mit dem Hauptprogramm an. Natürlich besitzt ein Nut/OS Programm auch eine main() Funktion! Wie bei jedem anderen Programm auch, ist dies der Einsprungpunkt und der Start der eigentlichen Anwendung. Nut/OS selbst erledigt zuvor bereits einiges an Initialisierung, darum muss sich der Anwender jedoch nicht kümmern. Eine Besonderheit gibt es dennoch: wie bereits erwähnt ist Nut/OS ein Multitasking Betriebssystem und setzt dabei auf kooperatives Multitasking. Das bedeutet, dass das eigentliche Hauptprogramm (also die main() Funktion) selbst bereits ein eigenständiger Thread ist. Dies wird am Ende der main() Funktion deutlich, wo das Programm in eine Endlosschleife läuft. Zuvor wird die Priorität des main-Threads auf die niedrigste Stufe gesetzt. Die eigentliche Arbeit erledigen später die Webserver Threads.

Das Hauptprogramm

```
int main(void)
{
     uint32 _ t baud = 115200;
     uint8 _ t i;
     int rc;

     uint8 _ t mac[] = DEFAULT _ MAC; /* MAC address that shall be assigned to the ethernet
     device */

     /* Initialize the GPIOs used to control the relais */
     GpioPinConfigSet(PORT _ RELAIS, PIN _ RELAIS1, GPIO _ CFG _ OUTPUT);
     GpioPinSet(PORT _ RELAIS, PIN _ RELAIS1, 0);

     GpioPinConfigSet(PORT _ RELAIS, PIN _ RELAIS2, GPIO _ CFG _ OUTPUT);
     GpioPinSet(PORT _ RELAIS, PIN _ RELAIS2, 0);
```

Die main() Funktion unter Nut/OS bekommt keine Parameter übergeben und ist daher als void deklariert. Im Folgenden werden einige Variablen definiert, die später benötigt werden. Interessant sind hier nur die Baudrate sowie die MAC Adresse. Diese ist im Programmkopf als Konstante hinterlegt. Dieses Beispiel verwendet im übrigen eine MAC Adresse, bei der das „lokal verwaltet" Flag gesetzt ist. Das bedeutet, dass diese Adresse nicht weltweit eindeutig ist und wir selber sicherstellen müssen, dass die MAC Adresse in unserem Netzwerk eindeutig ist.

Wenn man vor hat ein Produkt mit einem Netzwerk Interface zu vertreiben, sollte man sich auf jeden Fall einen entsprechenden Adressraum eindeutiger MAC Adressen registrieren lassen.

Nach der Variablendefinition werden als erstes die GPIO Pins, die mit unseren Relais verbunden sind initialisiert und auf low-Pegel gesetzt, so dass die Relais nach dem Einschalten zunächst einmal abfallen. Im nächsten Schritt werden die Peripherie-Schnittstellen Initialisiert...

```
/*
* Initialize the uart device.
*/
NutRegisterDevice(&DEV _ UART0, 0, 0);
freopen(DEV _ UART0 _ NAME, „w", stdout);
freopen(DEV _ UART0 _ NAME, „r", stdin);
_ ioctl( _ fileno(stdout), UART _ SETSPEED, &baud);

printf(„\n\neNet-sam7X Webserver demo... Nut/OS Version %s\r\n", NutVersionString());

/* Initialize the MMC card and mount the filesystem */
init _ mmc();
```

Treiber werden unter Nut/OS im Allgemeinen mit der Funktion NutRegisterDevice() registriert. DieseFunktion initialisiert die Schnittstelle und stellt sicher, dass unser Programm über den Device Descriptor bzw. den Device Namen (hier die konstante DEV_UART0_NAME) mit der Schnittstelle kommunizieren kann. In diesem Fall initialisieren wir die UART0, öffnen die Standard I/O File-Descriptoren auf dieser Schnittstelle und weisen Ihr per _ioctl() noch die Baudrate (hier 115200 Baud) zu. Und schon kann per printf() beliebiger Text ausgeben werden. So kann man leicht per Hyperterminal (Windows) oder Minicom (Linux) die Ausgaben des Programms verfolgen oder das Programm durch Eingaben steuern. Debugging wird so zum Kinderspiel.

An dieser Stelle wird es Zeit die MMC Karte zu initialisieren und das Dateisystem zu mounten.

```
/* Initialize the MMC card
and mount the filesystem */


init _ mmc();
```

Das Initialisieren der MMC Karte sowie das Mounten des FAT Datei Systems erledigt die Funktion init_mmc(). Diese Funktion stelle ich aus Platzgründen an dieser Stelle nicht im Detail dar, sie ist natürlich im Sourcecode der Demo vorhanden. Kurz zusammen gefasst greifen hier drei Komponenten ineinander. Der Dateisystem Treiber für das FAT Dateisystem (im Nut/OS Jargon PHAT Filesystem), der SPI-Bus Treiber für den physikalischen Zugriff auf die MMC Karte sowie schließlich der MMC Karten Treiber für den logischen Zugriff auf die Karte. Zunächst werden nun die Treiber initialisiert und miteinander verknüpft so dass anschließend das Dateisystem gemountet werden kann. Danach kann auf das Dateiesystem zugegriffen werden. Auch hier bietet Nut/OS eine sehr komfortable API, so dass wir einfach mit den Funktionen open / fopen / read / write usw. auf die Dateien auf der MMC Karte zugreifen können. Ganz so wie wir es vom PC gewöhnt sind.

Da Nut/OS mehrere unterschiedliche Dateisysteme gleichzeitig mounten kann, müssen wir beim Zugriff auf eine Datei den Pfad der Datei immer inklusive des Mountpoints angeben. So öffnet ein

```
fopen("PHAT0:/index.html",
"r+");
```

die Datei „index.html" Im Wurzelverzeichnis der MMC Karte.

Im nächsten Schritt wird das Ethernet Interface initialisiert.

```
do {
     rc = NutRegisterDevice(&DEV _ ETHER, 0, 0);
     if (rc != 0) {
             puts(„Registering ethernet device failed! Is a cable connected? Retrying...\r\n");
     }
} while (rc != 0);

printf(„Configuring %s... „, DEV _ ETHER _ NAME);

if (NutNetIfConfig2(DEV _ ETHER _ NAME, mac, inet _ addr(DEFAULT _ IPADDR),
inet _ addr(DEFAULT _ NETMASK), inet _ addr(DEFAULT _ GATEWAY)) == 0) {
     puts(„OK\r\n");
} else {
     puts(„Failed\r\n");
     NutSleep(500);
     while(1);
}
```

Nachdem das Ethernet Device erfolgreich mit `NutRegisterDevice()` registriert wurde, wird per `NutNetIfConfig2()` eine statische IP Adresse, eine Subnetz-Maske sowie ein Standard Gateway festgelegt. In unserem Beispiel verwenden wir folgende Einstellungen:

```
IP:           192.168.1.200
Netmask:      255.255.255.0
Gateway:      192.168.2.254
```

Die Einstellungen sind im Programm-Kopf als Konstante hinterlegt und können den eigenen Bedürfnisse angepasst werden. Natürlich könnte man die IP Adresse auch dynamisch per DHCP abfragen. Hier sei auf die diversen Beispiele verwiesen, die in dem Ethernut / Nut/OS Paket enthalten sind.

Damit unser Webserver jetzt auch weiß, wo die Webseiten zu finden sind, müssen wir den „Webroot“, also das Wurzelverzeichnis unseres Webservers registrieren. Die Konstante `HTTP _ ROOT` zeigt dabei wieder auf unsere MMC Karte: `PHAT0:/`

```
/* Register root path. */
printf(„Registering HTTP root ,“ HTTP _ ROOT „‘...“);
if (NutRegisterHttpRoot(HTTP _ ROOT)) {
     puts(„failed“);
     for (;;);
}
puts(„OK“);
```

Damit das Programm auf Eingaben / Steueranweisungen durch den Webbrowser des Benutzers reagieren kann, bedarf es noch einer Kommunikations-Schnittstelle mit dem Benutzer. In der Welt der Web- Anwendungen wird dies im Allgemeinen über CGI Funktionen erledigt. Auch unser Programm enthält ein CGI welches die beiden an das eNet-sam7X Modul angeschlossenen Relais steuert.

Im Gegensatz zu einem PC Webserver stellt ein CGI hier jedoch kein eigenes Programm dar sondern eine Funktion. Diese wird vom Webserver aufgerufen, wenn der User die entsprechende URL des CGIs im Webbrowser eingibt oder der Webbrowser Formulardaten an ein CGI schickt. Diese CGI Funktion muss vor Verwendung ebenfalls registriert werden. Registrieren bedeutet hier, dass der Funktion ein virtueller Name bzw. eine URL zugewiesen wird. Zuvor wird jedoch noch festgelegt, in welchem unter-Ordner des Webroots dieses CGI zur Verfügung gestellt werden soll.

```
/*
* Register the path where CGIs shall be located. This is a virtual path only
* as CGIs are virtual files too
*/

NutRegisterCgiBinPath(„cgi-bin/“);

/*
* Register some CGI samples, which display interesting
* system informations.
*/

NutRegisterCgi(„relais.cgi“, CGI _ Relais);
```

Unser CGI lässt sich jetzt also über die URL `http://192.168.1.200/cgi-bin/relais.cgi` ansprechen.

Als letztes bleibt nur noch den eigentlichen Webserver zu starten und anschließend das Hauptprogramm in einer Endlosschleife enden zu lassen. Aber Achtung: Wie bereits erwähnt nutzt Nut/OS ein kooperatives Multitasking. Beim kooperativen Multitasking findet ein Thread-Wechsel immer nur an definierten Punkten statt und es muss jederzeit sichergestellt sein, dass die CPU an einen anderen Thread abgegeben werden kann.

Eine einfache Endlosschleife würde die Programmausführung „einfangen“ und die CPU nie wieder für andere Threads freigeben. Ein einfaches `NutSleep()` hilft hier weiter und legt den Haupt-Thread an dieser Stelle schlafen.

Aber zurück zur eigentlichen Webserver Funktion. Diese ist als Thread implementiert. Damit wir mehrere HTTP Verbindungen auf einmal bearbeiten können starten wir diesen Thread gleich mehrfach.

```
/*
* Start the server threads.
*/
for (i = 0; i <= 8; i++) {
      char thname[] = „httpd0“;

      thname[5] = ‚0‘ + i;
      NutThreadCreate(thname, Service, (void *) (uintptr _ t) i, 4096);
}

printf(„Web server on %s:80 ready\r\n“, DEFAULT _ IPADDR);

/*
* We could do something useful here. In this case we are just setting
* the main thread to the lowest priority and are just waiting.
*/
NutThreadSetPriority(254);
for (;;) {
      NutSleep(60000);
}
return 0;
}
```

Der Server-Thread

Der eigentliche Webserver besteht aus dem Server-Thread. Dieser sorgt dafür, dass ein TCP Socket geöffnet wird und auf Port 80 auf eine eingehende Verbindung wartet. Dabei schläft der Thread so lange, bis der Webbrowser eine entsprechende Anfrage an den Webserver sendet.

Sobald eine Verbindung zustande gekommen ist, unser Webserver also ein „accept“ auf dem Socket ausgeführt hat, wird dem Socket ein Filedescriptor zugewiesen, der anschließend der Funktion „NutHttpProcessRequest()“ übergeben wird.

Diese Funktion ist das eigentliche Herzstück unseres Webservers. Sie ist eine fertig implementierte API Funktion von Nut/OS und sorgt dafür, dass die HTTP Anforderungen des Webbrowsers ausgewertet und die angeforderten Daten (Dateien oder dynamische Inhalte) an den Webbrowser ausgeliefert werden.

Zusammengefasst sieht der Code dazu wie folgt aus:

```
THREAD(Service, arg)
{
      TCPSOCKET *sock;
      FILE *stream;
      uint8 _ t id = (uint8 _ t) ((uintptr _ t) arg);

      for (;;) {
            if ((sock = NutTcpCreateSocket()) == 0) {
                  ...
            }

            NutTcpAccept(sock, 80);
            ...
            if ((stream = _ fdopen((int) ((uintptr _ t) sock), „r+b“)) == 0) {
                  printf(„[%u] Creating stream device failed\n“, id);
            }
```

```
            else {
                  NutHttpProcessRequest(stream);
                  fclose(stream);
            }
            NutTcpCloseSocket(sock);
      }
}
```

Aber wie kommen jetzt die Steueranweisungen des Benutzers zu den Relais?

Wir erinnern uns, dass im Hauptprogramm ein CGI registriert wurde. Dieses CGI wurde der URL `/cgi-bin/relais.cgi` zugewiesen. Die URL ist relativ zum Webroot des Servers zu lesen.

Hat der Webbrowser nun einen GET oder POST Request mit dieser URL an unseren Webserver gesendet, so erkennt die Funktion NutHttpProcessRequest(), dass es sich dabei um unser CGI handelt und ruft dieses auf. Dieses braucht die Anfrage dann nur noch auszuwerten und die Relais entsprechend zu setzen. So schließt sich der Kreis...

Das Relais CGI

Bei einem Webserver wird immer zwischen statischem Inhalt (HTML Dateien, Bildern etc.) sowie dynamischen Inhalt unterschieden. Ein CGI stellt die einfachste Form von dynamischem Inhalt dar. Es kann dabei entweder nur auf Eingaben reagieren oder eben auch vollständige HTML Seiten dynamisch generieren und an den Webbrowser ausliefern. In unserem Beispiel wollen wir jedoch nur auf Benutzereingaben reagieren und keine Ausgaben produzieren. Wie wir im folgenden Code Ausschnitt sehen, wird trotzdem mindestens ein HTTP Header zurück an den Webbrowser geschickt. Normalerweise mit dem Status Code „200 OK“. In unserem Fall senden wir jedoch ein „204 No content“ und signalisieren dem Browser damit, dass er auf keine Antwort vom CGI warten (und diese ggf. darstellen) soll.

Das CGI sieht (in gekürzter Form) wie folgt aus:

```
int CGI _ Relais(FILE * stream, REQUEST * req)
{
      NutHttpSendHeaderTop(stream, req, 204, „No content");
      NutHttpSendHeaderBottom(stream, req, „", -1);

if (req->req _ method == METHOD _ POST) {
      char *name;
      char *value;
      int i;
      int count;

      NutHttpProcessPostQuery(stream, req);
      count = NutHttpGetParameterCount(req);
      /* Extract count parameters. */
      for (i = 0; i < count; i++) {
            name = NutHttpGetParameterName(req, i);
            value = NutHttpGetParameterValue(req, i);

            if (strcmp(name, „set1") == 0) {
                  GpioPinSet(PORT _ RELAIS, PIN _ RELAIS1, 1);
                  printf(„=> Set relais 1\n");
            } else
            if (strcmp(name, „clear1") == 0) {
                  GpioPinSet(PORT _ RELAIS, PIN _ RELAIS1, 0);
                  printf(„=> Clear relais 1\n");
            } else

            ...

            }
      }

      fflush(stream);

      return 0;
}
```

Zunächst wird der HTTP Header zurück an den Browser geschickt. Anschließend überprüfen wir, ob es sich bei der Anfrage um einen POST Request handelt (GET Requests werden ignoriert) und werten ggf. die übergeben Parameter aus. Dabei wird nur der Parameter Name betrachtet. Klickt der Benutzer auf der Webseite zum Beispiel auf den „Set" Button von Relais 1, so wird der Parameter „set1" übertragen. Ein Einfacher strcmp() reicht zur Auswertung aus. Anschließend wird das entsprechende GPIO Pin entweder auf high (set) oder auf low (clear) gesetzt.

Und fertig ist unser Internet-Schalter!

Compilieren / Installieren

Jetzt bleibt nur noch übrig das Programm zu compilieren und anschließend auf das Board zu übertragen. Dem zum Download bereit stehenden Code liegt neben den Webseiten natürlich auch ein passendes Makefile bereit. Die Pfade im Makefile müssen ggf. der eigenen Nut/OS Installation angepasst werden. Anschließend reicht jedoch ein

```
make
```

auf der Kommandozeile um das Programm zu übersetzen. Es wird eine Datei „webserver.bin" erzeugt, die anschließend z.B. per JTAG in das Flash des Mikrocontrollers geschrieben werden kann. Hierfür eignet sich das OpenSource JTAG Debugging Tool „OpenOCD" hervorragend. Passende JTAG Adapter sind günstig (ab ca. 30 EUR) z.B. von Olimex (z.B. ARM-USB-Tiny) oder Egnite (Turtelizer 2) zu bekommen. Alternativ kann man natürlich auch das SAM-BA Tool von Atmel zum Programmieren des eNet-sam7X Boards über den USB Port dafür verwenden. Beide Methoden werden im eNet-sam7X Handbuch [8] detailliert anhand von Beispielen erklärt.

Zur Inbetriebnahme muss man nur noch eine Micro-SD Karte (Max 2GB, Fat formatiert) mit der Webseite und den zugehörigen Dateien bespielen und in den Sockel auf dem eNet-sam7X Modul einlegen, und schon ist der „Internet-Schalter" bereit seine Arbeit aufzunehmen.

Fazit / Ausblick

Im Rahmen dieses Artikels konnten wir natürlich nur ein ganz einfaches Beispiel konstruieren. Nichts desto trotz ist schon dieses einfache Beispiel eine vollständige und benutzbare Applikation. Ein über das Internet steuerbarer Schalter mit zwei eigenständigen Umschaltern...

Eine realistische Applikation könnte leicht ein deutlich umfangreicheres Benutzer Interface bieten, dynamische Webseiteninhalte einbinden und natürlich komplexe Steuerungs- und Regelungsanwendungen implementieren. Die Webserver Funktionalität alleine lastet die AT91SAM7XC512 CPU nicht einmal annähernd aus. Die meiste Zeit wartet das Programm ja schließlich auf Anfragen aus dem Internet. So bleibt hinreichend Rechenkapazität für eigene Ideen und Regelungs-Algorithmen übrig.

Ideen für weitere und ggf. komplexere Applikationen finden sich schnell und bleiben der Kreativität der Leser überlassen ;-)

Mein Kollege hat zum Beispiel mit einer nur unwesentlich komplexeren Schaltung sowie ein wenig Software eine vollständige Web-Basierte Regelung und Steuerung für eine Solar-Thermie Anlage gebaut. Nähere Infos inklusive Live-Anzeige der aktuellen Betriebsdaten können finden sich auf folgender Webseite:

http://solar.thermotemp.de/

Ein guter Startpunkt um vielleicht Ideen für eigene Projekte rund um das eNet-sam7X Modul zu entwickeln.

Eine weitere Idee wäre die Einbindung von Lua als embedded Scriptsprache. Da embedded Lua bereits Bestandteil der Nut/OS Umgebung ist lässt sich ein einfaches webbasiertes Programmier-Modell entwickeln, so dass auch Anwender Programme für das eNet-sam7X Modul entwickeln können, die bislang wenig Erfahrungen in der Mikrocontroller Welt haben und trotzdem alle Vorzüge eines embedded Systems nutzen wollen.

Quellenverzeichnis

[1] http://www.embedded-it.de/pdf_free/eNet-sam7X-schematic.pdf

[2] http://www.ethernut.de/

[3] htt p://www.ethernut.de/ de /firmware/index.html

[4] http://www.embedded-it.de/bsp/eCross.php

[5] http://www.yagarto.de

[6] http://www.embedded-it.de/bsp/eNet-sam7X_bsp.php

[7] http://www.embedded-it.de/bsp/demos/webserver_relais_demo.tar.gz

[8] http://www.embedded-it.de/pdf_free/eNet-sam7X-hardware-manual.pdf

[9] http://solar.thermotemp.de/

Wir bauen einen Rechner Teil 2

Vom Schaltplan übers Linux zum Lötofen

Hubert Högl <hubert.hoegl@hs-augsburg.de>

Die Platine

Im letzten Heft schrieb ich „Vermutlich könnte im März der Platinenplan fertig sein.“, was aus heutiger Sicht ziemlich daneben war. Das Routen der Platine stellte sich als schwierig heraus, weil wir zum einen alle Anschlüsse des i.MX auf Steckerleisten haben wollten und zum anderen ziemlich viele Vorgaben gemacht haben, wo die Bauteile und Steckverbinder sitzen sollen, ohne genau zu beachten, wo sich die Anschlüsse in der BGA Matrix des i.MX befinden. Deswegen gab es Ende April ein Treffen, bei dem wir die ursprüngliche Grösse der Platine (7 x 7 cm) und Platzierung der Bauteile (J1-J3 oben, J4-J6 unten) über den Haufen geworfen haben.

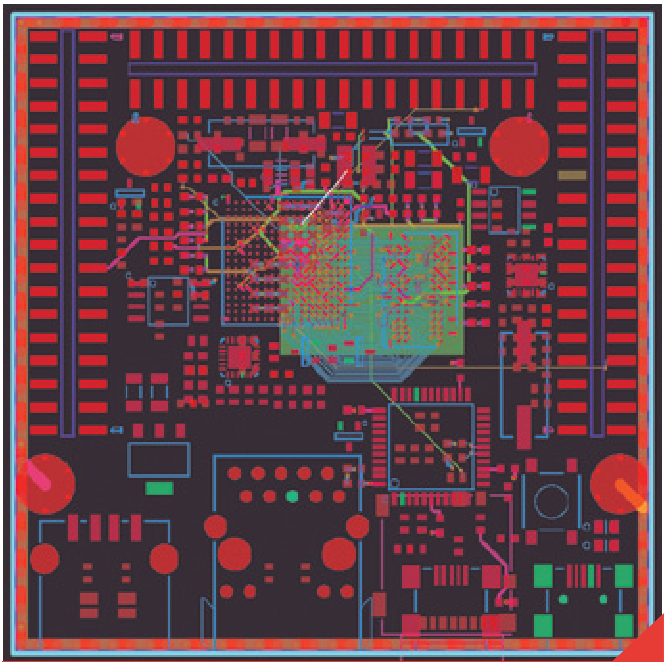


Abb. 1: KiCad Platine

Nun wird die Platine vermutlich 8 x 8 cm gross, oben sind zwei Stecker und unten vier. Die Bauteile sitzen nun fast alle auf einer Seite, so dass die Bestückung einfacher wird (vor allem wenn wir, wie angestrebt, unsere eigenen Lötversuche machen). Die Befestigungslöcher sind nun weiter nach aussen gewandert, damit die Bahnen besser Platz haben.

Ziemlich viel Zeit hat auch das probeweise Routen mit dem Freerouter [2] gekostet. Diese Anwendung ist in Java programmiert und läuft deshalb deutlich langsamer als kommerzielle Router wie Specctra oder Electra. Am Anfang haben wir darauf spekuliert, dass das Routen mit einem schnellen PC mit mehreren Kernen skaliert, das hat jedoch nicht funktioniert. Unser schnellster Testrechner war ein Core i7 960 mit 3,6 GHz, der jedoch beim Routen nur um den Faktor 1,3 schneller war als ein deutlich langsamerer Core i3. Als ungefähre Zeit zum Routen haben wir eine bis zwei Wochen veranschlagt. Die folgende Abbildung ist ein Screenshot eines Versuches des nun verworfenen Entwurfes mit dem Freerouter.

Aufbau der Prototypen

Angenommen, wir hätten bereits ein paar leere Platinen vom Platinenhersteller, dann würden wir in etwa so vorgehen:

Die Bauteile mit BGA Gehäuse (i.MX und SDRAM) würden wir am Anfang drauflöten. Ob das mit dem simplen „Pizzaofen“ geht, mit dem man gewöhnliche SMD Bauteile drauflöten kann, werden wir sehen. Benedikt hat mit dem Ofen, den man in der Abbildung sieht schon ein paar Monate Erfahrung im Auflöten von normalen SMD Bauteilen.

Er plant nun gerade einen Versuch, bei dem ein paar alte AVR32AP7000 Controller im BGA Gehäuse damit gelötet werden und danach die Qualität der Lötung mittels JTAG/Boundary Scan geprüft werden soll. Der AP7000 hat 16 x 16 Lötkugeln mit 1 mm Abstand. Mittlerweile hat er schon die BGA Testplatinen bekommen. Im nächsten Heft wird über diese Versuche ein Erfahrungsbericht stehen.

Falls das klappen sollte, würden wir dieses preiswerte Lötverfahren auch beim Gnublin Board einsetzen. Der i.MX28 hat 17 x 17 Lötkugeln mit 0.8 mm Abstand. Schwierig wird bei dieser Technik sicher auch die Platzierung der Bauteile sein, da



Abb. 2: Lötofen

wir kein optisches Platziergerät haben. Falls das Löten mit diesen einfachen Mitteln nicht funktioniert, dann würden wir es mit einer aufwendigeren Anlage probieren. An unserer Hochschule gibt es bei Professor Villain ein Labor für Löttechnik (siehe [5]). Die für uns wesentlichen Geräte wären ein optischer „Fineplacer“ und eine Dampfphasen Lötanlage. Daneben gibt es auch noch einen Siebdruckautomaten. Man sieht die Geräte unter dem angegebenen Link. Nachdem die BGA Gehäuse aufgelötet wären, würden wir die restlichen SMD Bauteile entweder im Pizzaofen oder von Hand auflöten. Wenn das Löten in zwei Phasen gemacht wird, dann bräuchten wir auch zwei Schablonen für die Lötpaste, eine für die BGA Bausteine, die andere für die restlichen Bauteile. Wie gut das funktioniert, wissen wir noch nicht. Neben diesen zwei gerade beschriebenen Möglichkeiten gäbe es auch noch eine dritte Möglichkeit: Es haben sich ein paar Leute gemeldet, die uns gerne mit kommerzieller Fertigungstechnik unterstützen würden. Ich würde es gut finden, wenn auch andere unabhängig von uns Prototypen aufbauen und testen würden, natürlich aber auf eigenes Risiko und Kosten.

Linux auf dem i.MX28 EVK

Wir haben in der Zwischenzeit das i.MX28 EVK von Freescale (das Geschenk) in Betrieb genommen. Es ist über RS-232 und Ethernet an einen kleinen Serverrechner angeschlossen (ein alter Transmeta Efficeon TM8000 mit 1 GHz), so dass man auch von aussen über SSH mit dem Board arbeiten kann. Dieser Rechner ist natürlich viel zu langsam um die aufwendigen Kompilierarbeiten für Kernel und Root-Filesystem zu erledigen. Man sollte daher die Kompilierung auf dem eigenen Rechner erledigen und dann nur noch die Resultate per scp auf den Server übertragen.

Als Boot-Quelle habe ich die SD/MMC Karte ausgewählt, so dass nach dem Einschalten oder Reset sofort das U-Boot startet. Alle weiteren Aktiviäten kann man dann von U-Boot aus erledigen. Zum Beispiel kann man Linux über das Netzwerk holen und das Root Filesystem über NFS mounten. Der Server sieht so aus: Abb. 3.

Wie man aus Erfahrung weiss, hängt sich der Embedded Linux Rechner beim Entwickeln gelegentlich auf, so dass nur noch das Drücken der Reset Taste hilft. Um auch diesen Vorgang fernsteuern zu können, habe ich eine Schaltung mit einem Atmel AVR Mega32 aufgebaut, die durch eine V-USB [3] Firmware über USB zum Ansteuern der Reset Leitung verwendet werden kann. Man sieht die kleine Lochrasterplatine in der Abbildung 4.

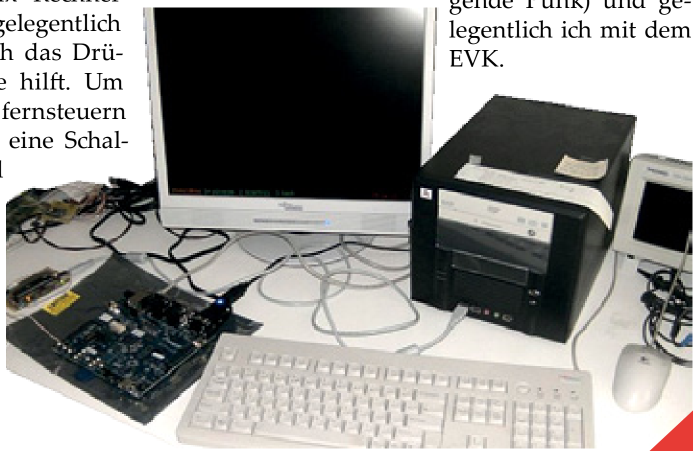


Abb. 3: Server mit EVK

Wer Experimente mit dem Board am Server machen möchte, dem gebe ich gerne die nötigen Daten zum Login. Zur Zeit arbeiten nur zwei Studenten (siehe folgende Punkt) und gelegentlich ich mit dem EVK.

Welches Linux hätten Sie denn gerne?

Standardmässig gibt es von Freescale die „LTIB“ Distribution, die zwar das Board unter Linux zum Leben erweckt, die aber von den meisten Embedded Linux Gurus eher skeptisch beäugt wird. Die Anwendergemeinde um LTIB scheint nicht besonders gross zu sein, so dass manche Pakete sehr alt sind. Der Python Interpreter, mit dem wir einiges vor haben, hat zum Beispiel die sehr alte Version 2.4.

Neben LTIB gibt es auch noch andere Cross-Bauumgebungen für Embedded Linux, etwa OpenEmbedded (OE) [6]. Andreas Müller hat OpenEmbedded für den i.MX28 angepasst, die Details findet man in der Mailingliste. OE wäre sicher die bessere Wahl im Vergleich zu LTIB, das Rahmenwerk ist sehr mächtig, jedoch auch schwieriger zu verstehen.

Wenn man die Rechenleistung des i.MX28 betrachtet, dann würde auch nichts im Wege stehen, „nativ“ auf dem Board zu arbeiten. Das heisst, man kompiliert Programme mit dem GCC direkt auf dem i.MX28 Controller. Mit 450 MHz Takt sollte man sogar relativ zügig arbeiten (kompilieren) können. Durch die weite Verbreitung von Geräten mit ARM Controllern unterstützen nun auch fast alle grossen Distributionen die ARM Architektur. Hier sind drei davon:

- Debian GNU/Linux auf ARM http://www.debian.org/ports/arm/index.de.html
- Gentoo Linux ARM Project http://www.gentoo.org/proj/en/base/arm/index.xml
- Fedora/ARM http://fedoraproject.org/wiki/Architectures/ARM

Da mich der native Betrieb bei der ARM Plattform stark interessiert, beschäftigen sich in diesem Sommersemester auch zwei Studenten mit dieser

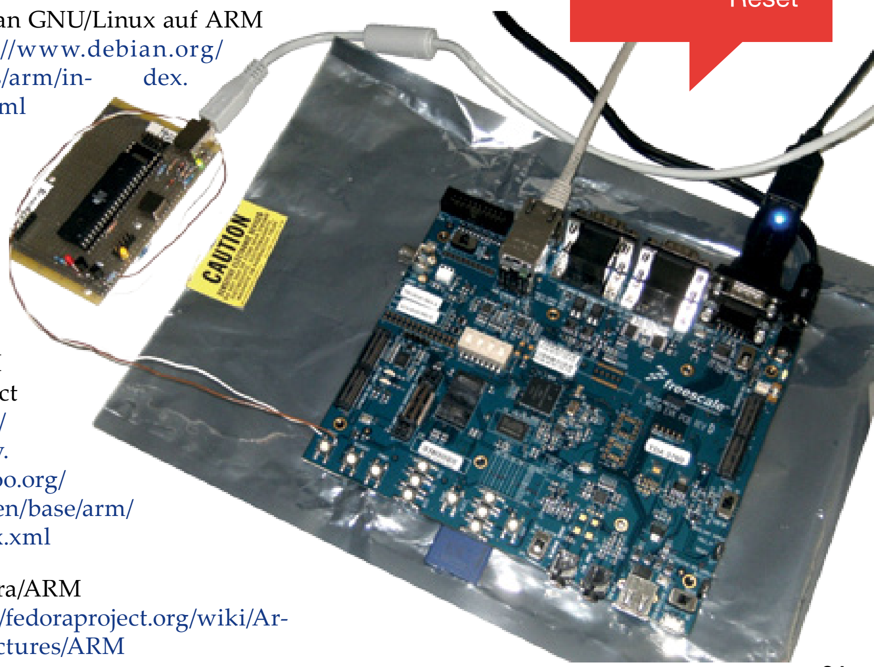


Abb. 4: Board mit Reset

Thematik. Ich hoffe durch diese Unterstützung bald zeigen zu können, wie gut sich die grossen Distibutionen für ARM eignen. Ein vergleichbares Board, das auch diesen Weg geht, ist das FOX G20 [7]. Es verwendet Debian/ARM.

Weitere Baustellen

Ein wesentliches Ziel des Projektes ist es, Anfängern ein preiswertes Board in die Hand zu geben, um Experimente mit Embedded Linux zu machen. Natürlich muss es dazu auch einen Platz im World Wide Web geben, an dem Anleitungen zu finden sind in denen steht wie man mit dem Board umgeht. Vermutlich hängt es ganz stark von der Qualität dieser Anleitungen ab, wie gross die Akzeptanz des Boards ist.

Damit sich das Ganze zu Lehrzwecken eignet, stelle ich mir vor, dass es eine Reihe von Versuchen gibt, die man ohne grosse Probleme nachvollziehen kann. Jeder Versuch soll genau die Pins der Gnublin Erweiterungsstecker zeigen, an denen man einfache externe Schaltungen anschliessen kann, die man auch selber im Labor aufbauen kann.

Hier sind einige Beispiele:

- Taster und LED an GPIO Pins anschliessen und über gpio/sysfs ansteuern. Die Abbildung 5 zeigt eine mit Kupferlackdraht verdrahtete Lochrasterplatine. Auch die „Flugdrähte" sind mit Einzelkontakten und Schrumpfschlauch selbst gemacht:
- PWM Ausgang ansteuern und damit LED dimmen.
- Analogeingang einlesen.
- Einen freien UART in Betrieb nehmen und per Programm mit einer Gegenseite kommunizieren, z.B. einem PC mit USB-zu-UART Adapter.
- SPI Schnittstelle in Betrieb nehmen (MISO und MOSI verbinden).
- „DOG" Display (16x2 LCD) über SPI ansteuern.

Diese (und weitere) Experimente mache ich bereits in meiner „Embedded Linux" Veranstaltung an der Hochschule Augsburg, allerdings auf dem NGW100. Wer hier gute Ideen für weitere Versuche hat, kann das gerne schreiben.

Um die bei den meisten Versuchen erforderlichen Anschlüsse der Erweiterungsstecker deutlich zu zeigen, schwebt mir eine dynamische Technik vor, bei der man den Platinengrundriss im SVG Format im Browser anzeigt und je nach Versuch bestimmte Stecker und Pins über JavaScript farblich hervorhebt. Die

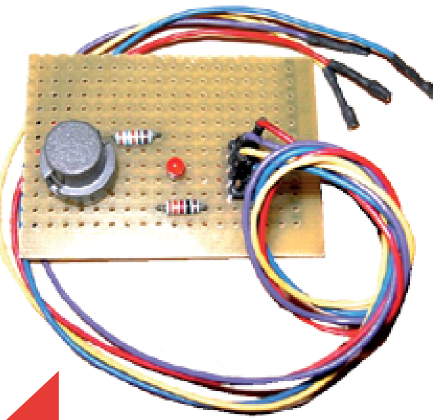


Abb. 5: GPIO

Information darüber, was hervorzuheben ist, wird vom Webserver über eine Board-Repräsentation geliefert, die in Python geschrieben ist. Sogar das Zeichnen des SVG Grundrisses kann man mit Python erledigen (mit dem pysvg Modul). Aktuell beschäftigen sich ein paar Studenten mit dieser Technik

Name und Logo

Obwohl es nicht so wichtig ist, beschäftigt mich immer noch die Suche nach einem passenden Namen und einem Logo. Zur Zeit verwenden wir Benedikt's ersten Vorschlag Gnublin.

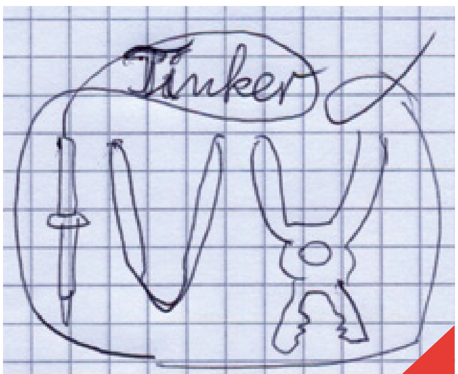


Abb. 6: Logo Vorschlag

Mir kam vor einiger Zeit auch noch die Idee TinkerTux oder TinkerBoard, nachdem ich den Artikel [8] las. Das Lexikon dict.leo.org sagt, dass „to tinker" herumbasteln bedeutet. Ich habe mal eine Skizze für ein Logo aufgemalt, „TUX" soll aus Lötkolben, Pinzette und Zange sein:

Wahrscheinlich ist der Vorschlag vollkommen ungeeignet, da gute Logos oft sehr elementar sind, zum Beispiel die liegende Acht mit Plus und Minus beim Arduino Projekt. Wer von Logos was versteht darf sich gerne bei mir melden!

Vielleicht sollten wir uns auch von der neuen Mode anstecken lassen und Tiernamen verwenden, so wie das mit Beagle, Panda und Hawk schon gemacht wird. Welches Tier würde zu uns passen?

Michael Ilg hat übrigens „StuBux" vorgeschlagen, ausgeschrieben „Studenten-Board für Embedded-Linux".

Ich freue mich auch weiterhin über Eure Vorschläge.

Links

[1] EPJ Nr. 8, „Wir bauen einen Rechner für Embedded Linux!" http://journal.embedded-projects.net

[2] Freerouter http://www.freerouting.net

[3] V-USB http://www.obdev.at/products/vusb

[4] Bcontrol http://elk.informatik.fh-augsburg.de/gnublin-cdrom/bcontrol/

[5] Professor Villain an der Hochschule Augsburg (Bereich Löttechnik) http://www.hs-augsburg.de/~villain/Laborzimmer/Loettechnik/Loettechnik.html

[6] OpenEmbedded http://www.openembedded.org

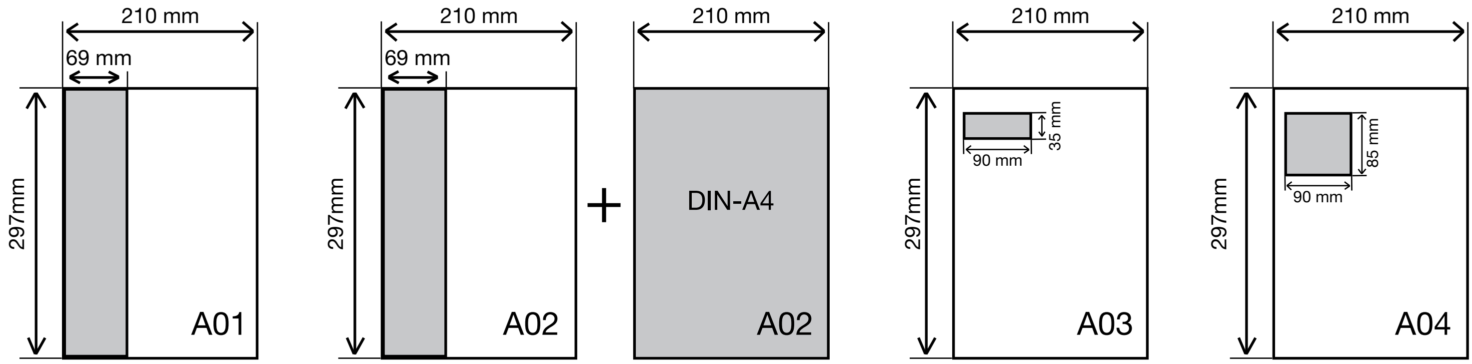
[7] FOX G20 http://www.acmesystems.it

[8] Anand Santhanam and Vishal Kulkami, Linux system development on an embedded device. Tinkering with PDAs for fun and profit, 2002. http://www.ibm.com/developerworks/library/l-embdev/index.html

Ihre Anzeige im embedded projects Journal

Ansprechpartner:
embedded projects GmbH
Holzbachstraße 4
D-86152 Augsburg
Tel. +49(0)821 / 279599-0
Fax: +49(0)821 / 279599-20
Mail: journal@embedded-projects.net

A01: 1/3 Seite **A02: Bundle** **A03: Jahressponsor** **A04: Würfel**



embedded projects Journal:

Das embedded projects Journal ist eine Zeitschrift, die unter einer Open-Source Lizenz steht. Alle Artikel und die Zeitschrift können so unter Einhaltung dieser Lizenz weiter verarbeitet werden. Das besondere an dem Open-Source Projekt Journal ist, dass die Zeitschrift nicht nur frei als PDF-Version verfügbar ist, sondern von den Lesern kostenlos abonniert werden kann. Druck und Porto werden durch Anzeigen von Sponsoren finanziert.

Aktuell wird jede Ausgabe ca. 8000 mal online und ca. 2000 mal als Papierversion gelesen.

Anzeigenformate und Preisliste:

Bezeichnung	Bezeichnung	Format in mm Breite x Höhe	Preis*	Erscheinung in
A01	1/3 Seite	69 x 297 mm	150 €	1 Ausgabe
A02	Bundle	69 x 299 mm + 210 x 297 mm	400 €	1 Ausgabe
A03	Jahressponsor **	90 x 35 mm	100 €	4 Ausgaben
A04	Würfel	90 x 85 mm	100 €	1 Ausgabe

* Alle Preise zzgl. Ust.
** Im Angebot inbegriffen: 1. Frei Haus Lieferung jeder Ausgabe.
2. Ihre Anzeigen wir zusätzlich auf unserer Homepage unter: http://www.ep-journal.de im Firmenverzeichnis veröffentlicht.

Ausgabe	Anzeigenschluss	Erscheinungsmonat
03 / 2011	15.08.2011	September
04 / 2011	15.11.2011	Dezember
01 / 2012	15.02.2012	März
02 / 2012	15.05.2012	Juni
03 / 2012	15.08.2012	September
04 / 2012	15.11.2012	Dezember



Bestellfax an +49(0)821 / 279599-20

☐ Anzeige in nächster Ausgabe veröffentlichen
☐ A01 ☐ A02 ☐ A03 ☐ A04
☐ Anzeige veröffentlichen in / ab Ausgabe ___ / ___

Firma: ______________________
Vor- / Nachname: ______________________
Anschrift: ______________________
____
Tel. / E-Mail: ______________________

Interesse an einer Anzeige?

info@embedded-projects.net

Auf Mitarbeiter-Suche?

Dann könnte dies Ihr Platz im nächsten Journal sein

journal.embedded-projects.net/stellenmarkt

embedded - projects.net
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Werdet aktiv!

Das Motto: Von der Community für die Community !

Das Magazin ist ein Open Source Projekt.

Falls du Lust hast, Dich an der Zeitschrift durch einen Beitrag zu beteiligen, würden wir uns darüber sehr freuen. Schreibe deine Idee an:

sauter@embedded-projects.net

Regelmäßig

Die Zeitschrift wird über mehrere Kanäle verteilt. Der erste Kanal ist der Download als PDF - Datei. Alle Ausgaben sind auf der Internetseite [1] verfügbar. Diejenigen, die lieber eine Papierversion erhalten möchten, können den zweiten Kanal wählen. Man kann sich dort auf einer Internetseite [1] in eine Liste für die gesponserten Abos eintragen. Beim Erscheinen einer neuen Ausgabe wird dank Sponsorengeldern an jeden auf der Liste eine Ausgabe des aktuellen Journal versendet. Falls man den Versandtermin verpasst hat, kann man das Heft auch zum Preis von einem Euro über einen Online - Shop [2] beziehen.

[1] Internetseite (Anmeldeformular gesponserte Abos): http://journal.embedded-projects.net

[2] Online - Shop für Journal:
http://www.embedded-projects.net

Bitte frei machen

embedded projects GmbH
Holzbachstraße 4
D - 86152 Augsburg

Name / Firma

Straße / Hausnummer

PLZ / Ort

Email / Telefon / Fax

○ Ich möchte jede zukünftige Ausgabe erhalten

○ Wir möchten als Hochschule / Ausbildungsbetrieb jede weitere Ausgabe bekommen. Bitte gewünschte Anzahl der Hefte pro Ausgabe ankreuzen. (5) (10)

○ Ich möchten im embedded projects Journal werben oder eine Stellenanzeige aufgeben. Bitte schicken Sie mir Infomaterial, Preisliste etc. zu.

Sponsoren gesucht!

Damit wir weiterhin diese Zeitschrift für jeden frei bereitstellen können, suchen wir dringend Sponsoren für Werbe- und Stellenanzeigen. Bei Interesse meldet Euch bitte unter folgender Telefonnummer: 0821 / 2795990 oder sendet eine E-Mail an die oben genannte Adresse.

Impressum

embedded projects GmbH
Holzbachstraße 4
D-86152 Augsburg
Telefon: +49(0)821 / 279599-0
Telefax: +49(0)821 / 279599-20

Veröffentlichung: 4x / Jahr
Ausgabenformat: PDF / Print
Auflagen Print: 2500 Stk.
Einzelverkaufspreis: 1 €

Layout / Satz: EP
Titelfoto: Claudia Sauter